DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

(1)

AFIT/GOR/ENS/91M-12

A PROTOTYPE META-LANGUAGE AND AUTOMATED TRANSLATOR FOR DECISION ANALYSIS PROBLEM FORMULATION

THESIS

Jerry R. Puyear
Captain, USAF

AFIT/GOR/ENS/91M-12

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
March 1991

3. REPORT TYPE AND DATES COVERED
Master's Thesis

4. TITLE AND SUBTITLE
A PROTOTYPE META-LANGUAGE AND AUTOMATED TRANSLATOR FOR DECISION ANALYSIS PROBLEM FORMULATION

5. FUNDING NUMBERS

6. AUTHOR(S)
Jerry R. Puyear, Captain, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GOR/ENS/91M-12

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This thesis research effort develops a prototype meta- language for decision analysis problem description and automated software to translate the meta-language into a format useable by decision analysis software solvers. MELADA, the meta-language developed, makes use of special symbols to provide a succinct language for the automation environment yet provides the means for user comprehension. Complete syntax and implementation rules are included. The MELADA translator software developed, DAT, stores the data given by MELADA in a file using a prototype standard format, DASF, designed to hold all required data to solve a problem. DAT provides error checking and a message output file. TREESOLVER, an add on program to solve decision trees stored in DASF format was designed. The program uses the external DASF file directly to bypass size limitations of internal memory. All software is written in Turbo Pascal 5.5, is compatible with any MS-DOS computer system and is free to government users in source code format, including a user's manual. Finally, several example problems are solved using MELADA, DAT, DASF, and TREESOLVER to validate their viability.

14. SUBJECT TERMS
(2) # Decision Analysis, Decision Theory, Decision Tree, Influence Diagram, Decision Analysis / Software

15. NUMBER OF PAGES
82

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

AFIT/GOR/ENS/91M-12

A PROTOTYPE META-LANGUAGE AND AUTOMATED TRANSLATOR FOR DECISION ANALYSIS PROBLEM FORMULATION

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science (Operations Research)

Jerry R. Puyear,
Captain, USAF

March, 1991

THESIS APPROVAL

STUDENT: Captain Jerry R. Puyear CLASS: GOR-91M

THESIS TITLE: A Prototype Meta-Language and Automated Translator for Decision Analysis Problem Description

DEFENSE DATE: 20 February 1991

COMMITTEE:	NAME/DEPARTMENT	SIGNATURE
Advisor	Major Bruce W. Morlan/ENS	[illegible]
Reader	Major Ken W. Bauer/ENS	Kenneth W Bauer

Preface

This thesis provides the beginning of the standardization and automation of decision analysis problem description and computer input. This effort was made on behalf of my thesis advisor, Major Bruce Morlan, and anyone who has spent an eternity at the computer keyboard trying to input data to make use of a "time and effort saver" software package. If it saves anyone just a fraction of the time I spent at the computer terminal working on this thesis, it will be worth all the effort.

I would like to thank Major Morlan for suggesting this topic and allowing me to wander but never completely stray from my chosen task. I also want to thank my reader, Major Bauer who was game enough to take on this task even though he was busy advising several of my classmates.

Finally, I give my biggest thanks to my wife, Donna, and daughters Christine and Renee, for their love and effort to get by while I was preoccupied with what they must think was the world's longest computer game.

Jerry R. Puyear

Table of Contents

List of Figures

Figure Page

List of Tables

AFIT/GOR/ENS/91M-12

Abstract

The major goals of this thesis research project were to prepare a prototype meta-language for decision analysis problem description and provide a software package to automatically translate a problem from a meta-language ASCII file into a format that can be used by decision analysis software solvers.

MELADA, a meta-language, was developed for decision analysis problem description/formulation. It makes use of special symbols to provide a succint language for the automation environment yet allows for additional information to be added to keep the data readable by the human user. Complete syntax and emplementation rules are included.

DAT, a decision analysis translator was also developed. It provides an automated means of storing a MELADA problem discription into an external storage file using a prototype standard format, DASF, designed to hold all required data to solve a problem. It provides error checking and a message output file. Written in Turbo Pascal 5.5, it is compatible with any MS-DOS computer system and the source code and users manual are free to government users.

In addition, TREESOLVER, an add on program to solve decision trees stored in DASF format was designed. The program uses the external DASF file directly to bypass size limitations of internal memory. Like DAT, it is written in Turbo Pascal and comes in source code format.

Finally, several example problems are solved using MELADA, DAT, DASF, and TREESOLVER to validate their viability.

A PROTOTYPE META-LANGUAGE AND AUTOMATED TRANSLATOR FOR DECISION ANALYSIS PROBLEM FORMULATION

I. Introduction

1.1 Background

Decision analysis is a methodology to apply logic and preference to problems with uncertainty in order to provide a rational approach to problem solving. This methodology can be broken into three phases: a deterministic phase in which the structuring of a model of the problem takes place, a probabilistic phase in which probabilities are assigned to the uncertainties in the model to arrive at an "optimum" solution, and an informational phase in which a cost/benefit analysis of additional information to reduce uncertainty is made (5:585-587). Viewed in a mathematical sense, these three phases are analogous to formulation of the problem equation, selection of values for the equation variables to obtain the solution, and running sensitivity analysis on the variables.

Formulation methods in the deterministic phase are as varied as in the mathematical field. For example, a graphical formulation of the problem can be developed using either the familiar decision tree or the influence diagram, "a recently developed graphical modeling tool for representing both the conceptual elements and the mathematical structure of a problem" (2:1). If the problem is limited in size and a graphical view is not desired, a decision table may be used instead. The selection of a method to use is currently based on personal preference or the requirement of a chosen software package.

The computer software packages designed to automate the decision analysis methodology also vary in approach. Some of these packages process decision trees. Others, like AFids, work with influence diagrams (2:8). However, the packages do have some common elements. First, input is by user interaction which can require a large amount of time at a terminal and a knowledge of decision trees or influence diagrams in order to input the problem correctly into the computer. Second, each package requires that all problem information fit within internal memory which limits the size of problem that can be handled. Third, the packages do provide the ability to store problem information in external files but each package has its own file format. Finally, each package can "solve" the problem given the appropriate inputs.

1.2 Specific Research Objectives

Decision analysis methodology would benefit by having the ability to describe and store a problem in common terms and format from which conversion to one of the various formulation methods or software packages could easily be made. Therefore, the four objectives of this research project were:

1. Develop a prototype meta-language to provide a standard means of describing decision making problems under uncertainty involving only discrete variables.
2. Build a user friendly software package that translates a meta-language text file into a prototype standard format input file for decision analysis optimizing software packages.
3. Develop a software package that uses the decision tree method to optimize decision making problems stored in a standard format input file.
4. Provide application examples to validate the meta-language and software developed.

Each objective has an intended goal. The goal of the first objective is to have a meta-language that is easy to write, read, comprehend, and, most importantly, can efficiently express information and concepts required to solve decision making problems by computer. The goal of the second objective is to have a high level computer language based translator that will provide error checking and the associated messages to aid the user in preparing an error free, complete meta-language input file for translation. The translated file should hold the needed information in a format that will allow the design of add on programs to convert from the translated file format to the specific format needed for current software optimizers. The format should also be designed to encourage direct use of the file as an input format for future software programs. The goal of the third objective is to show that the translated storage file can be used by optimizing software. The goal of the last objective is simply to show that a meta-language and translator are viable for the decision analysis field.

II. Review of Literature

2.1 Introduction

This review will cover the underlying concepts, format, and terminology used with three decision analysis model representations: the decision table, the decision tree, and the influence diagram. It will also review automated encoding efforts and current software in the decision analysis field.

2.2 Concepts

2.2.1 Basic concepts. According to Pratt, Raiffa, and Schlaifer decision making under uncertainty is based on the assumptions that a decision maker will be able to quantify preferences between outcomes, quantify beliefs in the possibility of events occurring, and provide preferences that follow the consistency rules of transitivity and substitution. From this base, they show that a decision maker can make a choice by computing an index for each option and then selecting the option with the largest index. They define the index as

$$\prod \equiv \sum_i P(E_i)\pi(c_i)$$

where E_i is in a set of mutually exclusive events $\{E_1, ..., E_n\}$, $\pi(c_i)$ is a scaled preference for consequence c_i, and $P(E_i)$ is a scaled measure of belief in event E_i happening. In addition, Pratt, Raiffa, and Schlaifer suggest you could call the scaled measure of belief a probability (since it acts like one), the scaled preference a utility, and the computed index an expected utility. They show that by following the given choice rule the decision maker will maximize his expected utility (13:35-40).

The suggestions of probability and utility by Pratt, Raiffa, and Schlaifer are not unique. The suggestion to call the scaled measure of belief a probability since it acts like one is given further credence by Lindley who states and shows in his article that probability is the only scaled measure of uncertainty that makes sense (8:1). What Pratt, Raiffa, and Schlaifer suggest could be called utility is based on assumptions that are contained in the axioms of current expected utility theory. In his article on expected utility (EU), Schoemaker lists the five axioms given by John von Neumann and Oskar Morgenstern that infer the existence of numerical utilities. The first and third axioms, as listed by Schoemaker, state people can make preferences, those preferences are transitive, and the principle of substitution holds (15:531). Obviously, those axioms are identical to the assumptions of Pratt, Raiffa, and Schlaifer.

The use of probability and expected utility theory by each of the three decision analysis models is documented by several authors. However, variations in terminology and theory exist. A breakdown by method follows.

2.2.2 Decision table. When the decision table is used to analyze a problem, the optimal selection is made by either minimizing expected oppoıtunity loss (EOL) or maximizing expected monetary value (EMV) (1:617-619)(14:8-9,27-29)(10:6). A term not restricted to monetary value, maximum expected value (MEV), is used by Moskowitz and Wright in place of EMV (11:123).

It should be noted that some of the authors consider the decision table limited in use. The decision matrix is considered equivalent to the decision tree according to Morris, but he thinks the tree is better because it depicts the order of the decision process (10:8). Anderson, Sweeney, and Williams suggest that the best use of the table for analysis is for problems having a limited size (1:619). Moskowitz and Wright also agree with the problem size limitation, stating that "... complex problems are very difficult to represent clearly in tabular (matrix) form" (11:132).

2.2.3 Decision tree. Analysis of the tree is accomplished through backwards induction, a process which relies on the principle of substitution, maximizing expected value, and the use of Bayes' Theorem to provide the appropriate probabilities (14:21-29).

2.2.4 Influence diagram. According to Baron, solving influence diagrams is accomplished by using a node reduction algorithm, first developed by Shachter, that is based on the maximizing expected value principle and Bayes' Theorem (2:12-15). Baron further states that he improved on the solution efficiency and problem size handling capability by implementing the Tatman and Shachter separable value function solving algorithm using dynamic programming (2:16-21,31).

2.3 Format

2.3.1 Decison table. The format of the decision table appeared to be standard in the works of several authors (10:6) (1:613)(11:117) except for one minor variation which exchanges the left side (alternatives or actions) and the top row (states of nature or states) (14:7). An example decision table format is shown in Table 2.1.

Table 2.1. Decision Table Format Example

option/state	low demand	high demand
Sell now	100	200
Sell later	50	25
Do not sell	-10	-25

2.3.2 Decision tree. The decision tree format, like the decision table format, appears to follow a standard convention in all the works reviewed. The convention uses square nodes to represent decision points, circle nodes to represent chance outcome points, and lines (branches) to represent each possible decision alternative and chance outcome. Branches also connect the nodes from left to right across the tree to show order (precedence). The far right hand side of the tree terminates with values or payoffs that result from reaching that particular terminal branch (1:619-

621)(11:130-132) (14:10-13)(10:7-8)(9:49-50). An example tree is shown in Figure 2.1.

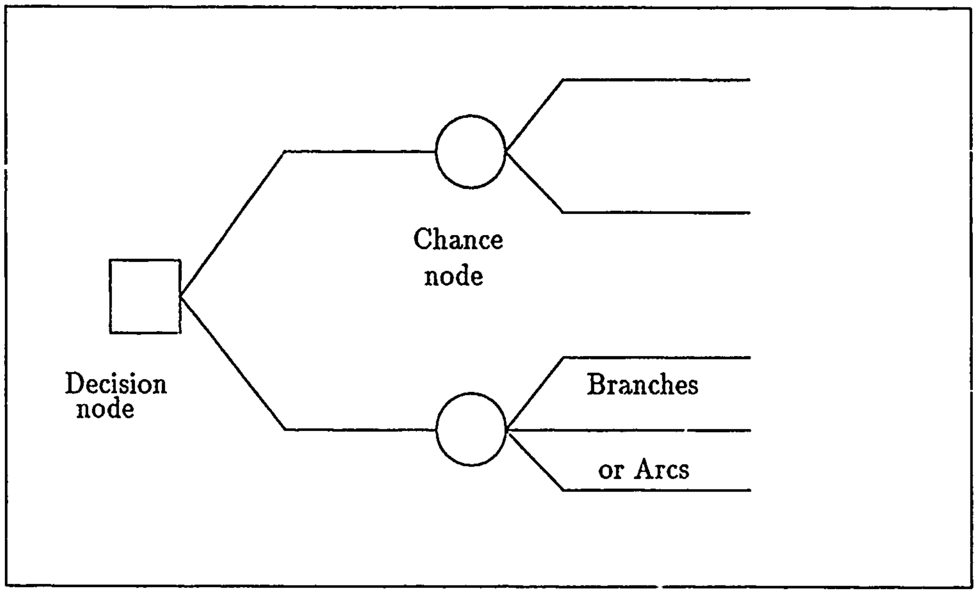


Figure 2.1. Example Decision Tree

2.3.3 Influence Diagrams. Influence diagrams have an acyclic graph format consisting of four types of nodes (decision, chance, deterministic, value) connected by arcs indicating informational flow and probabilistic dependence between the connected nodes (2:2). An example influence diagram depicting all possible components is shown in Figure 2.2. The following concepts are used in constructing the diagram according to Howard and Matheson:

1. Arcs may always be added between nodes (unless they create a cycle) because they only represent possible dependence.
2. Lack of an arc between nodes shows their independence.
3. An arc may be reversed in direction as long as both nodes involved have the same set of information (same immediate predecessors).

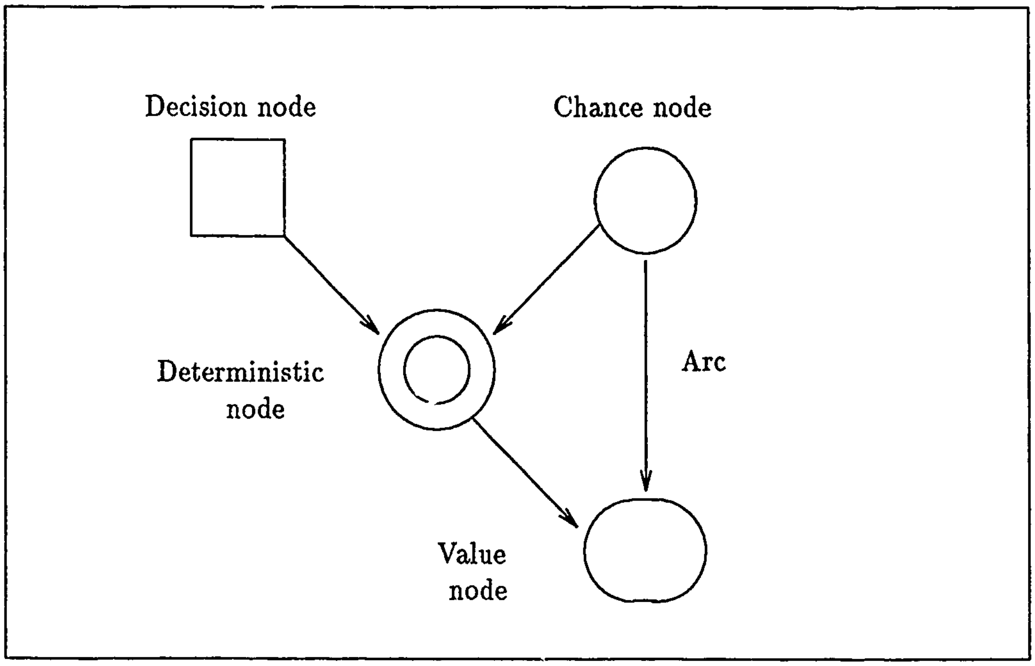


Figure 2.2. Example Influence Diagram

Howard and Matheson also point out that an influence diagram representation of a problem is not unique and that nodes do not need to be ordered nor do they depend on all predecessors like in decision trees. Further, they state that many influence diagrams can be converted into decision trees, but not all influence diagrams have corresponding trees (6:732-740).

2.4 Terminology

2.4.1 Decision table. Terminology to describe the decision table, unlike the format, was not as standard. Morris calls the table a matrix formulation that has alternatives down the left, "possible futures" (possible future states of the world) across the top, and an event at the intersection between the two (10:6). Anderson, Sweeney, and Williams describe it as a payoff table having alternatives like Morris, but use states of nature across the top and consider the intersection of the two to be payoffs (1:613). According to Moskowitz and Wright, the decision table is a payoff or decision matrix having actions on the left side, events across the top, and consequences at the action-event intersection (11:117).

2.4.2 Decision tree. Like the decision table, the terminology to describe the tree varies. Raiffa calls the tree a decision-flow diagram or tree while Moskowitz and Wright use tree diagram and decision tree, but both say it consists of decision and chance forks (Raiffa also calls the forks junctures, nodes or branching points), branches, and terminal payoffs at the right side of the tree (14:10-11)(11:130-131). Morris prefers chance node, event node, or probability node to chance fork. He further defines branches extending from decision nodes as alternative branches and those out of chance nodes as outcome branches. In addition, Morris says the tree terminates with terminal states (the branches on the right side of the tree that have a single payoff, value or outcome rather than a follow on node). (10:8). Anderson, Sweeney, and Williams also use nodes and branches, but replace chance with state-of-nature to describe the type of node or branch in the decision tree (1:619-620).

2.4.3 Influence diagram. Influence diagram terminology also has some minor variations. Owen uses state variable node (12:766) and Shachter uses probabilistic variable (16:590) rather than chance node like Howard, Matheson, and Baron (6:735)(2:2). Baron also breaks from the Howard and Matheson terms of informational influence and conditional influence by using informational flow and probabilistic dependence. In addition, Baron describes the node connectors as "arcs" while Howard and Matheson use "arrows" (2:2)(6:735).

2.5 Automated Encoding and Current Software

2.5.1 Automated encoding. The difficulty of entering problems into computers for solving has been addressed in several areas. Fourer, Gay, and Kernighan believe that the difficulty of translation can be reduced by the use of a "modeling language" and mention two languages (GAMS by Bisschop, Mearans, Brooke, and Kendrick in 1982 & 1988; MGG by Simons in 1987) developed for problem translation in linear programming (3:520). According to Ligeza, it is fairly common within the decision support field to use a logic-lii language readable by the decision maker to encode knowledge and Ligeza provides hi. own if-then-conditional format prototype language as an example (7:107).

2.5.2 Software. Current decision tree processing software packages include ARBORIST a general purpose commercial package by Texas Instruments (17:iii) and SUPERTREE, a package by Strategic Decisions Group (9). ARBORIST represents the typical capability of decision tree software: problem size limited by the amount of available internal memory, building of the tree by interactive entry of information through the keyboard using menus, and varying levels of data storage of designed trees for later retrieval. Personal experience with ARBORIST painfully teaches one that not all information is retained in long term storage. For example, probabilities and values are not retained unless entered as variables which means they must be entered again each time the file is retrieved. SUPERTREE, unlike ARBORIST, improves the input capability some by the ability to get partial input from popular spreadsheet programs (9:255-261).

Current influence diagram software packages include PerForma, a very basic package for small problems written by Burwell and Tatman; DAVID, a Duke University commercial package written by Shachter that has the most analysis features available according to Baron; and AFids, a package using the latest advances written by Baron (2:vii,8-9). AFids, the newest influence diagram package in the decision analysis field, uses the keyboard to create new influence diagrams via menus and the point and click method, but diagrams already created and stored in a file may be read in (2:52). AFids also relies on internal memory for storage of the problem which limits problem size (2:25).

2.6 Conclusions

Several conclusions can be made about a meta-language for decision analysis. First, the use of a meta-language to aid automated encoding is recognized as beneficial and is being pursued in other fields. Second, the decision analysis field has not concerned itself with development of a meta-language, but has stayed the course with nonstandard, interactive encoding of problems based on the graphical method used. Finally, the three methods discussed in the review appear to have some common analysis concepts and terms upon which a meta-language might be based.

III. MELADA: Meta-Language for Decision Analysis

This chapter discusses the meta-language designed to meet the first objective of this thesis, providing a standard way to describe decision making problems under uncertainty.

3.1 Requirements

Several requirements exist for any meta-language designed to deal with decision making under uncertainty. The language must be able to convey all the information needed to apply the probability and expected utility concepts described in the literature review. That means the language must be able to describe probabilities, utilities (values), decisions, decision alternatives, chance events, chance outcomes, deterministic nodes, value nodes, and the relationship between them. In addition, the terms must be unambiguous and succinct enough for computer use. Finally it requires a straightforward syntax that can be easily understood by both user and computer.

3.2 Terminology of Concepts

Selection of terms to describe the concepts was based on use within the decision analysis field, commonality between the three methodologies, and current use or ability to use in computer software. Frequency of use of a term within the decision analysis field, noted during the literature review, provided a list of the most popular terms for some of the concepts. Shown in Table 3.1 is a listing of the most popular terms used to describe decision making elements, elements whose outcomes are chosen randomly by nature, and worth to the decision maker of these particular elements.

Terms selected for the meta-language to describe several of the conceptual elements and their associated definition are listed below.

Table 3.1. List of Popular Terms

Decision analysis concept	Most popular terms for concept
Decision maker (DM) chooses course	Decision
Options DM can choose from	branches, alternatives, actions
Nature randomly chooses course	chance, state of nature
Options nature chooses from	branches, outcome, states of nature
Decision maker's worth of concepts	value, payoff

Decision A choice to be made from a list of alternatives by the decision maker.

Chance An event where nature randomly selects the outcome from among the available possibilities.

Alternative One of the possible options that the decision maker may choose from for a particular decision.

Outcome One of the possible results of a chance event.

Probability A numerical measure of belief that a particular outcome of a chance event will occur.

Payoff The worth (utility) to the decision maker of a particular combination of decisions, or outcomes stated in numerical terms.

Value Node A node in the influence diagram that represents the objective function (2:3).

Deterministic Node A node in the influence diagram whose value is totally determined by a combination of the preceding nodes' values (2:3).

There are additional terms that are not directly needed by the meta-language but can be standardized to help problem description. (To help clarify the following descriptions, meta-language elements are in boldface and the suggested additional standardized terms are in italics.) To standardize decision tree and influence diagram graphical description, a **decision** or **chance** should be called a *node* while an **alternative** or **outcome** should be called an *arc.* Decision tables should have **alternatives** on the left and **outcomes** listed along the top of the table. Because influence diagrams have **value nodes**, the value (utility) of an individual *node* or *arc* should be called a **payoff** to avoid confusion. Dependence or relational order terms are not required in the meta-language because it is conveyed by the order of the various elements within each sentence. However, *predecessor* and *successor*, commonly used to describe order, are used in the translator software and should be used as the standard terms.

3.3 Employment of Concepts

The specific information and relationships conveyed by MELADA to achieve the requirements mentioned above are categorized below.

- Identify distinctly each decision and chance.
- Identify distinctly each alternative and outcome.
- Link alternatives to their associated decision.
- Link outcomes to their associated chance node.
- Link successor decision and chance nodes to their predecessor alternative and outcome arcs.
- Assign payoffs to outcomes and alternatives that do not have successors.
- Assign probabilities to outcomes.
- Assign functions to value and deterministic nodes.

To accomplish these requirements in an automated environment some rules are necessary to reduce computational problems. These rules reduce the options the computer must handle, avoid the input of information prior to having the element to which the data belongs, and ensures the entering of all required data. Violation of the rules can result in an incorrect problem description or confusion as to what is correct by an individual looking at the meta-language description. The implementation rules are listed below.

1. Each identifier used must be distinct.
2. Assignment of probabilities or payoffs to an alternative or outcome can only be made after the alternative or outcome arc is associated with its parent decision or chance node.
3. Each alternative or outcome must be assigned either a follow on chance node, a follow on decision node, or a payoff.
4. Every outcome must be assigned a probability.
5. A probability or payoff can only be changed by correcting the original assignment sentence, not by overriding it with another assignment sentence.

Rule number two is required because the distinction of arc type (alternative or outcome) in the meta-language is based upon the type of parent node (decision or chance) and the fact that the information that can be assigned to an arc depends on its type. Rule number five is not meant as a limitation but is used to help the user avoid accidental changing of values. Since the meta-language is written to a flat ASCII file the correction of an assignment statement will be fairly easy.

It should be noted that deterministic nodes are converted to chance nodes by influence diagram software (2:33) and thus can be represented by chance nodes during input without the need for deterministic nodes. Also, the meta- language deterministic and value nodes were designed with string function inputs like AFids uses. The AFids method was selected for two reasons. First, AFids is noncommercial software and available at AFIT which makes it a logical choice to use as the primary influence diagram software. Second, AFids uses dynamic programming in its optimizing process and provides several state of the art advances for influence diagram optimizers (2:33).

3.4 Syntax

MELADA uses selected symbols to delimit the conceptual elements mentioned earlier in a sentence structure that conveys none, one, or more than one of the categorized requirements listed above. The delimiting symbols key the required data in each sentence from the optional information used to increase comprehension. Bracketed within the delimiting symbols are specific identifiers consisting of up to twenty characters selected by the user, special relational symbols, payoff values expressed as real numbers by the user, and probabilities unique to the problem input by the user. The delimiting symbols, special relational symbols, and special character combinations used in MELADA are listed below with a definition of what they represent and an example as appropriate. Items in italics are the ones the user chooses, those in boldface are required by the syntax.

[ ] Delimits a decision identifier— [*takeumbrella?*].

( ) Delimits a chance identifier— (*rain?*).

{ } Delimits an alternative or outcome— {*yes*}.

* * Delimits a value node identifier— **objective**.

Delimits a deterministic node identifier— #*drilling costs*#.

<> Delimits payoff values or probabilities— < *10.0* >.

Pr Probability of, used in front of chance or outcome identifiers when assigning probabilities— **Pr**(*rain?*).

= Used within a decision or chance element to link a decision or chance with one of its alternatives or outcomes. For example, (*rain?=yes*) means one outcome of chance node rain? is yes.

, Used to separate multiple entries or identifiers within payoff, probability, alternative, and outcome elements. For example, {*yes,no*} are two outcomes or alternatives.

" " Delimits a long definition of a particular identifier— "*anything the user puts within the double quotes is part of the definition* ".

% % Delimits a function string for value and deterministic nodes. Anything input by the user between the delimiting symbols is part of the function.

. Marks the end of a MELADA sentence.

When a probability element **Pr** is used in a sentence the payoff element $< payoff >$ that follows is defined as holding probabilities and not payoffs. Multiple alternative and outcome identifiers, payoffs, and probabilities are allowed in certain cases to reduce the sentences needed to input the data. Those cases are the linking of alternatives and outcomes to their parent node, the assignment of all outcome probabilities to a chance node, and the assignment of payoffs to several alternatives of a decision node or several outcomes of a chance node. A complete syntax chart of the MELADA elements is in Figure A.1, located in the users manual in Appendix A along with instructions for its use.

The order of the above elements within the MELADA sentence have special meaning so the sentence structure is limited to those element combinations needed to convey the information required. Some of the legal combinations and what they convey is listed below using proper delimiters and the standardized term as the element identifier ([*decision*] is a decision element).

[decision] {alternative(s)} Assigns alternatives to the given decision.

[decision]$< payoff >$ Assigns a worth (utility) to each alternative of the decision.

[decision] "definition" Provides a long definition of the given decision identifier.

(chance=outcome) [decision] Assigns outcome to chance if not already assigned and assigns decision as a follow on node to outcome.

(chance=outcome) $< payoff >$ Assigns a worth (utility) to the outcome and assigns outcome to chance if not previously done.

Pr(chance) < *probability* > Assigns a probability to each outcome assigned to chance.

Pr(chance=outcome) < *probability* > Assigns a probability to outcome and assigns outcome to chance if not already assigned.

Pr{outcome} < *probability* > Assigns a probability to outcome.

{outcome} < *payoff* > Assigns worth (utility) to outcome.

{outcome} [decision] Assigns decision as a follow on node to outcome.

{outcome} (chance) Assigns chance as a follow on node to outcome.

{outcome} "definition" Assigns a long definition to the outcome identifier.

*** value * % function string%** Assigns a function to the given value node.

#deterministic# %function string% Assigns a function to the given deterministic node.

Additional legal sentences identical to the ones above that have a decision element first can be made by substituting in a chance element for the decision element. Likewise, the compound chance element **(chance= outcome)** can be replaced with a compound decision element [*decision* = *alternative*] and **{outcome}** can be replaced with an **{alternative}** element to make additional sentences.

It should be noted that the sentences above show only the key elements and their required order within the sentence for clarity. Additional information can be added before, between, and after the elements as long as the information does contain character strings that could be considered elements according to syntax rules. A chart of the complete sentence structure syntax is in Figure A.2, located in the users manual in Appendix A along with the instructions for its use.

3.5 Limitations.

It must be noted that the ability to use the meta-language to transfer problems between influence diagrams and decision trees will be limited. Not all influence diagrams have counterpart decision trees (6:740) due to the fact that influence diagrams are not only used to solve decision making problems but to describe the elements and relationships of a problem without worrying about the mathematical aspects (2:1). For those problems that can be expressed by both methods the difference in format must be overcome. Influence diagrams hide the alternative and outcome data within each decision and chance node instead of displaying them like decision trees. Influence diagrams also allow for more than one predecessor or successor, unlike the decision tree. However, the data required remains the same. For example, each outcome of the influence diagram chance node must have a probability value for each predecessor's input (a conditional probability) just as each chance node in the decision tree is conditioned on its predecessor. The only difference is that the decision tree uses multiple chance nodes to represent the same uncertainty situation based on a different predecessor (condition). The difference in format between decision trees and influence diagrams for a simple problem is shown in Figure 3.1. Thus, to transfer from one method to the other requires the expansion or contraction of the nodes and the ability to realize which nodes are identical except for their predecessor (conditional given) input. MELADA does not have a direct means of showing those relationships at the present time. However, problems that result in symmetric trees could be transferred since the nodes at each level are the same and could be transformed into one influence diagram chance node or vice versa by a conversion package.

3.6 Employment of MELADA in an Automated Environment

This section explains how MELADA fits into the automated environment and its relationship to the software developed in this thesis effort.

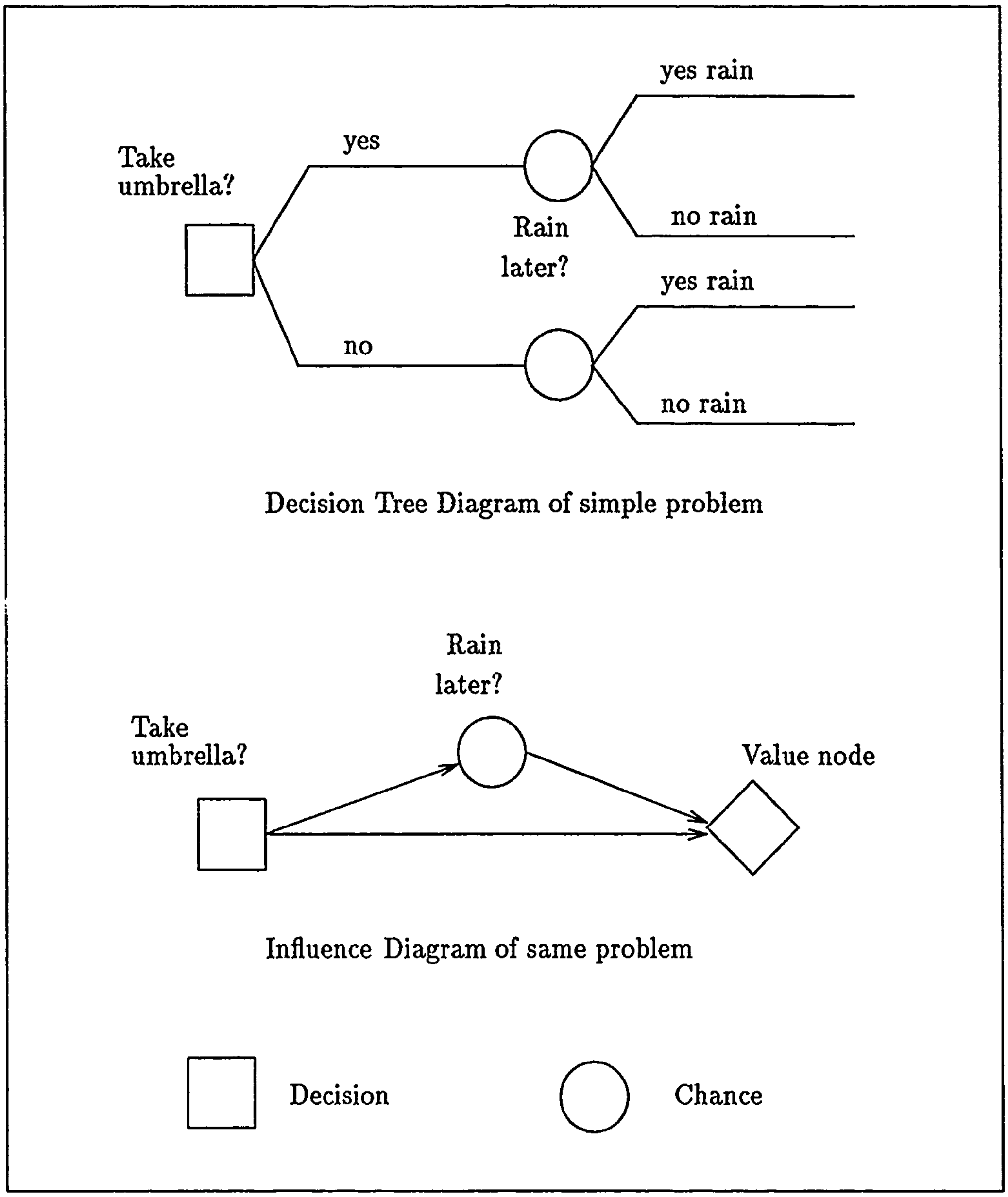


Figure 3.1. Example of Difference in Format

MELADA is designed to be the standard language used to build a flat ASCII input file that describes a decision making problem for the automated environment. The file can be built by the user with any word processor that makes ASCII files. This input file is then used by DAT, the decision analysis translator described in the next chapter, to build a file with a standardized format for holding only necessary problem information. The standardized storage file is then used as an input file for optimizing software, like TREESOLVER, the optimizing software package developed during this thesis effort and described in a later chapter. Figure 3.2 shows the progression from the user building a MELADA file to receiving the optimal results and the relationship between MELADA and the software.

In the following chapters the software packages just mentioned are described in greater detail to include their use of MELADA, program philosophy, and implementation instructions required to achieve the automated environment given in Figure 3.2. In addition, example applications are provided demonstrating the entire cycle from MELADA to optimum solutions.

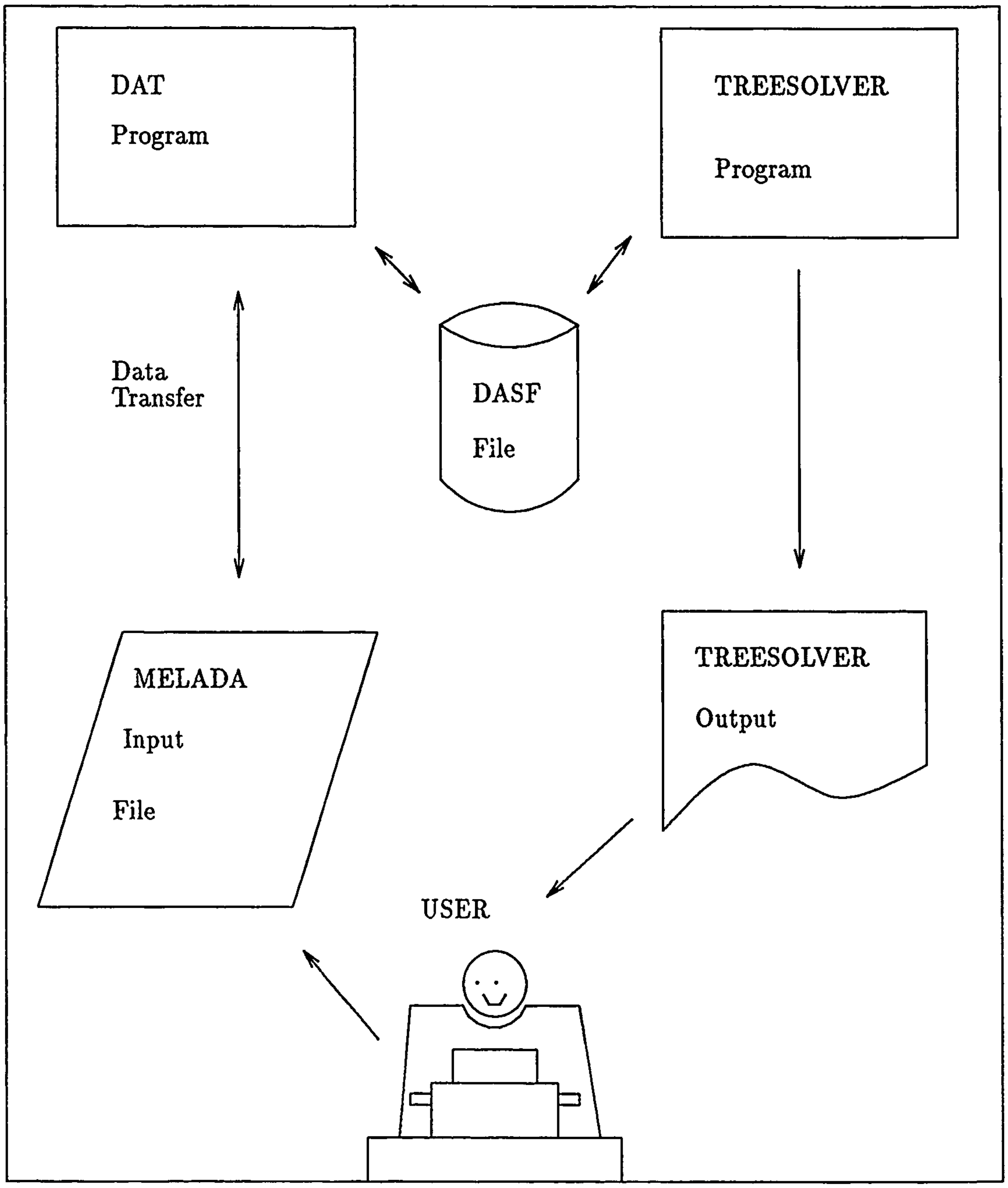


Figure 3.2. Automated Environment for Decision Analysis

IV. DAT: Decision Analysis Translator

This chapter discusses the software package designed to achieve the second thesis objective of providing an automated means of translating decision analysis problems in MELADA ASCII files into a standard format file for use by optimizing software packages.

4.1 Requirements

To achieve the second objective, the proposed translator must be able to meet several requirements. These requirements are:

- Parse an ASCII file written in MELADA.
- Provide error checking to find:
    1. incorrect syntax
    2. duplicate identifier entries
    3. data assigned to undefined elements
    4. elements assigned incompatible data
    5. duplicate data inputs (changes to data)
    6. probabilities outside of values $\{0, ..., 1\}$
    7. predecessor/successor loops
    8. probabilities for each chance node sum to 1.0
    9. missing data needed to solve a problem
- Provide a file of error messages.
- Provide a file of long element definitions given.
- Store the data in an file using a standardized format.
- Be easy to use, update or expand.

4.2 Program Philosophy

This section discusses the programing concepts and tradeoffs used in DAT to meet the meet the requirements set forth above. The areas of discussion include the parsing, error handling, storage of information, and the programming language used by DAT.

4.2.1 Parsing. DAT uses a character input rather than a line input from the MELADA file while parsing each MELADA sentence, the basic MELADA data block that is examined to find the special characters delimiting key elements to set them apart from optional data. This allows for a straightforward approach to parsing MELADA sentences, even when they are longer than one line or written in paragraph format. While this method requires more reads from the MELADA file, it eliminates the problems encountered with a character string. Those problems include keeping a pointer to remember at what position in the string the computer is 'looking', removing the key data by getting subsets from the string, and knowing when to refill the string.

One of the checks made during parsing is to see if the identifier just encountered already exists or is a new one. This check requires searching through the records to check for an identifier match. To help speed up the search which in the worst case (new identifier) must check every existing record, DAT begins the search from the latest entry and searches toward the first entry. If the person writing the MELADA file follows a logical tree progression then the identifier in question, should it exist, is more apt to be in the latest records stored. Therefore, it will be found with fewer comparisons than would be the case if the search started with the first record.

4.2.2 Error handling. Most of the error checking is accomplished during input to allow for giving the error location within the MELADA file. DAT counts the number of sentences processed and if an error is found during input, a descriptive error message containing the sentence number in which the error occurs is printed to a message file. Once an error is encountered the remaining MELADA assignments of that sentence are not carried out to prevent further errors. However, processing is continued, thus providing the user with feedback on all attempted sentences, rather than just through the first erroneous sentence. Only the check for missing data and the check to see that each chance node's probability sums to one occurs after input of all information. These two have error messages which give the identifier name of the node or arc containing the error.

4.2.3 Storage. The parsed information is stored during processing into a DASF (*D*ecision *A*nalysis *S*tandardized *F*ormat) file (discussed in detail in the next section). This eliminates the internal memory size limitation on a problem and allows for problems limited only by the available external storage capacity. The tradeoff for size is speed of processing since random access to the file must be made to read and write data into the records. However, this program was intended to allow for processing of large problems on a PC based system and speed was not considered to be a big requirement. To speed up the system at the cost of a relatively large maximum size for the problems that can be handled, a RAM virtual disk can be set up to hold the DASF file. Another method is to use higher speed hard drives, especially those with cache systems (basically a buffer system) that allow for optimizing access by keeping the latest and most used information in RAM.

4.2.3.1 Program language and structure. The program is written in Turbo Pascal, a common high level language that is easy to update and provides many user-friendly options. The Turbo Pascal record and object structures make updates to the storage record format simple to accomplish should the need arise. In addition, Turbo Pascal is very expandable with its unit structure which will help facilitate the design of programs to convert the DASF file into a required input format of current software. To help toward that end, DAT's procedures and functions were designed and put into units for easy incorporation into conversion programs.

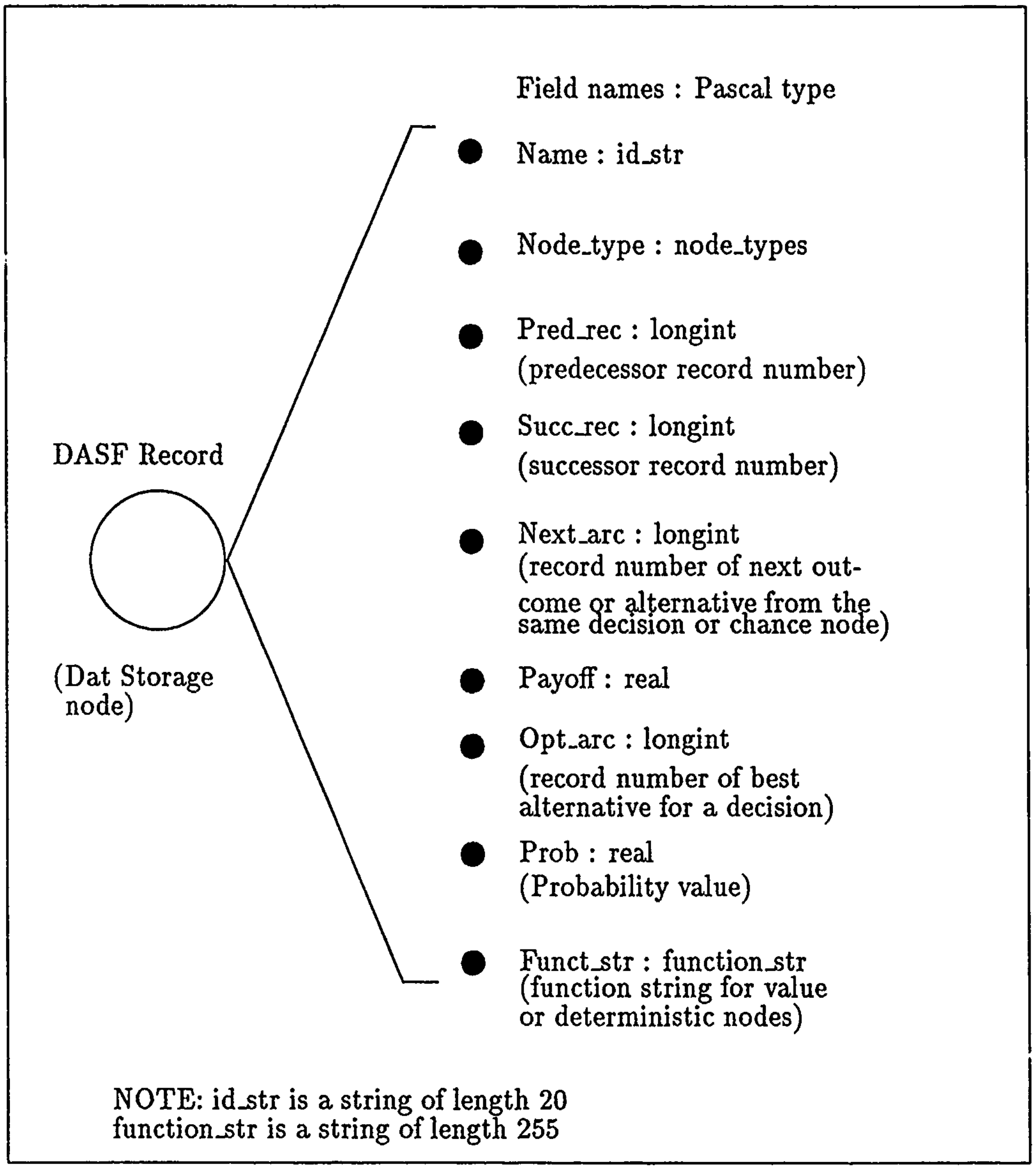
Every procedure is included in the *DATPRO.TPU* unit, except Readsentence, which has the unit called *READSENT.TPU* all to itself. *DATPRO.TPU* contains all the procedures that read and write the records in the standard format storage file and is the unit needed by any conversion program. The functions are in a unit called *DATFUN.TPU* and all the global variables are in a unit called *GLOBALS.TPU.* The source code is in files with the same name but have the .PAS extension. The source files include documentation for each procedure and function stating what it does, what procedures and functions it calls, and what global variables, if any, it uses. Additional in-line documentation exists within the procedures and functions to aid programmers. Appendix C contains an alphabetical summary of each procedure and function to include its use, input variables, and output variables.

4.3 DASF: Decision Analysis Standard Storage Format

A key to providing automated input and flexibility of choice between the various software packages available is a standard input format. If the input format is standardized, every new software program can be designed to use the one format. Current programs can have conversion programs designed to convert data from the standard format to the input format required or be updated to accept the format directly.

The decision analysis standardized format, DASF, makes use of the storage node record format used by DAT. It builds one record in the file for each storage node DAT builds. Each storage node contains the required information on one node or arc in the problem and has a standard format itself. The information included in each storage node and thus, each record in the DASF file, is shown in Figure 4.1. The use of the Pascal record structure makes the addition of new data requirements as simple as adding another field. Those fields not required by the particular element being described are given null or set values that reduce the space required in the file.

The individual records in the file are stored in the order that DAT builds the



- Name : id_str
- Node_type : node_types
- Pred_rec : longint (predecessor record number)
- Succ_rec : longint (successor record number)
- Next_arc : longint (record number of next outcome or alternative from the same decision or chance node)
- Payoff : real
- Opt_arc : longint (record number of best alternative for a decision)
- Prob : real (Probability value)
- Funct_str : function_str (function string for value or deterministic nodes)

NOTE: id_str is a string of length 20
function_str is a string of length 255

Figure 4.1. Data in DASF Record

storag. nodes, which is based on the order that the data is given in the MELADA file. The first record (record number zero) contains the problem identifier in the *name* field, the record number of the root node in the *succ_rec* field, and the number of the last record in the file in the *next_arc* field. To provide the logical order of the records, record numbers are used to point to the predecessors and successors. Therefore, the records are logically linked together in a tree format using a parent, child and right sibling approach. Figure 4.2 shows the link from one node using the parent, child, and right sibling approach. The parent is the record number contained in *pred_rec.*

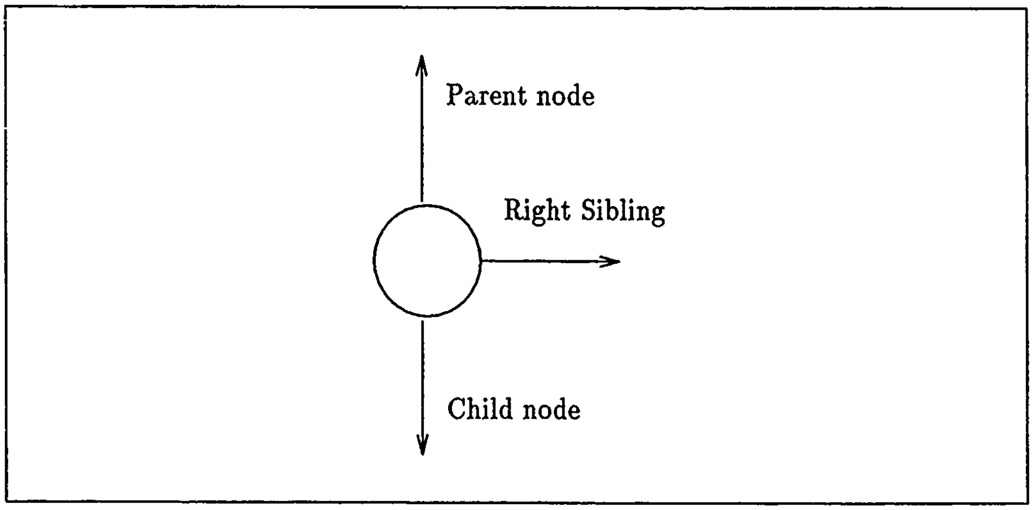


Figure 4.2. Parent, Child, Right Sibling Linkage Example

The child is the record number stored in *succ_rec* and the right sibling, used to link all the alternatives or outcomes of a particular decision or chance node together, is stored in *next_arc.* While the decision or chance node is only linked directly to its first (left most) alternative or outcome, each alternative and outcome can directly access the parent decision or chance node via the record number stored in *pred_rec* . The logical tree format that the computer 'sees' for the umbrella example shown in Figure 3.1 is given in Figure 4.3.

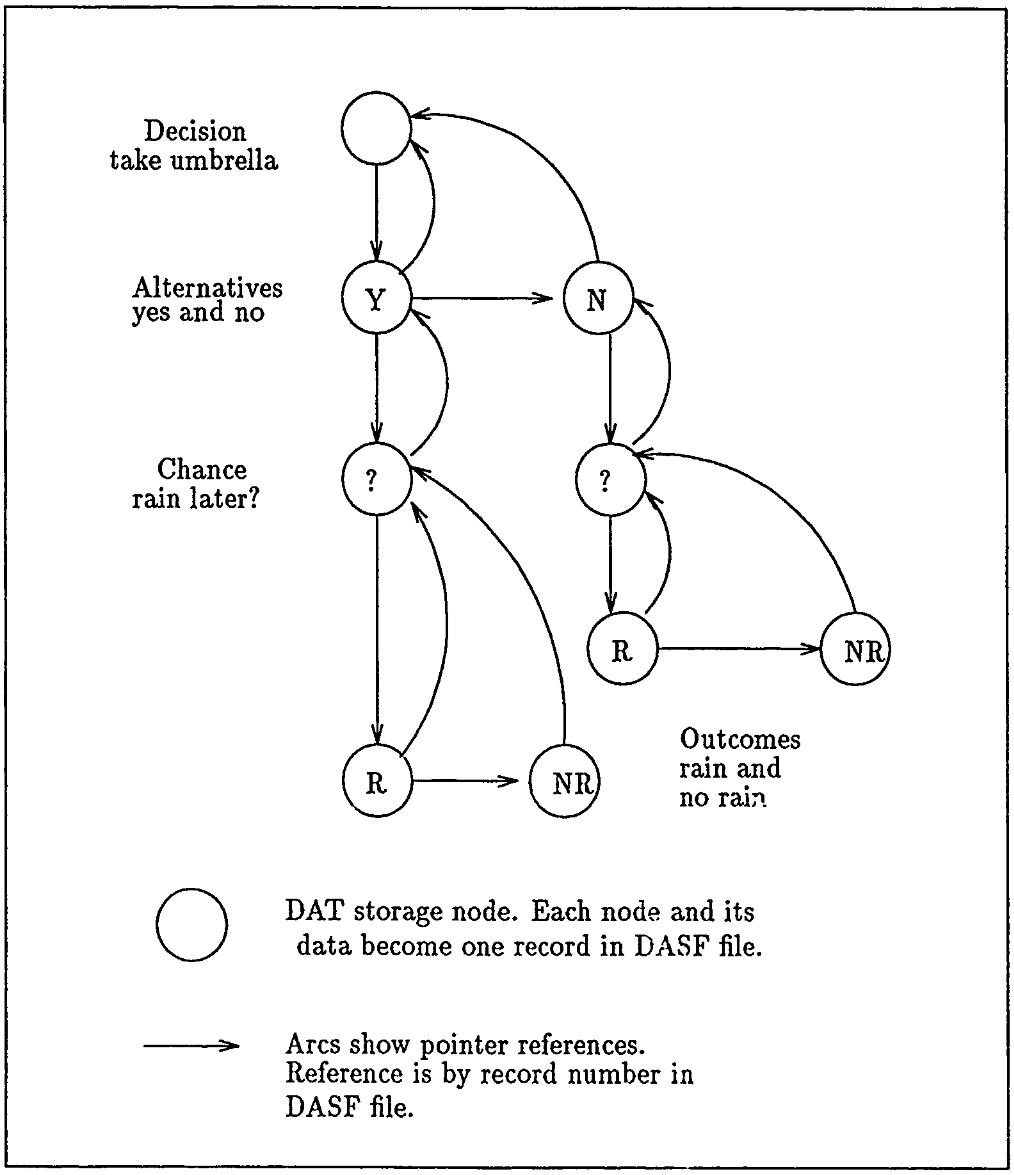

Decision
take umbrella
Alternatives
yes and no
Y
N
Chance
rain later?
?
?
R
NR
R
NR
Outcomes
rain and
no rain
DAT storage node. Each node and its data become one record in DASF file.
Arcs show pointer references. Reference is by record number in DASF file.

Figure 4.3. DASF Logical Record Tree

4.4 Implementation

DAT is a stand alone program that requires two input files and builds three output files during execution. One input file is the ASCII MELADA file with a problem name or identifier written on the first line of the file. DAT uses a line read to get the problem identifier so it must be the only item on the first line. The problem name can be as long as the line but only the first 20 characters are kept. Leading spaces will count as part of the name. The other input file is a small text file providing DAT the drive, path, and name of the MELADA input file and the three output files, each on a separate line. The file must be named *Filesfor.dat* and placed in the same directory as DAT in order for DAT to find it. The file provides the flexibility to name the input and output files as desired without changing the DAT source code and recompiling it every time. The three output files are the message file which contains the error messages and processing remarks, the definition file which holds the long definitions or explanations of the shorter identifiers used for each node and arc, and the actual DASF output file which holds all the problem information. Figure 4.4 shows the relationship between DAT and the files it uses.

Instructions for using DAT are in the users manual provided in Appendix A.

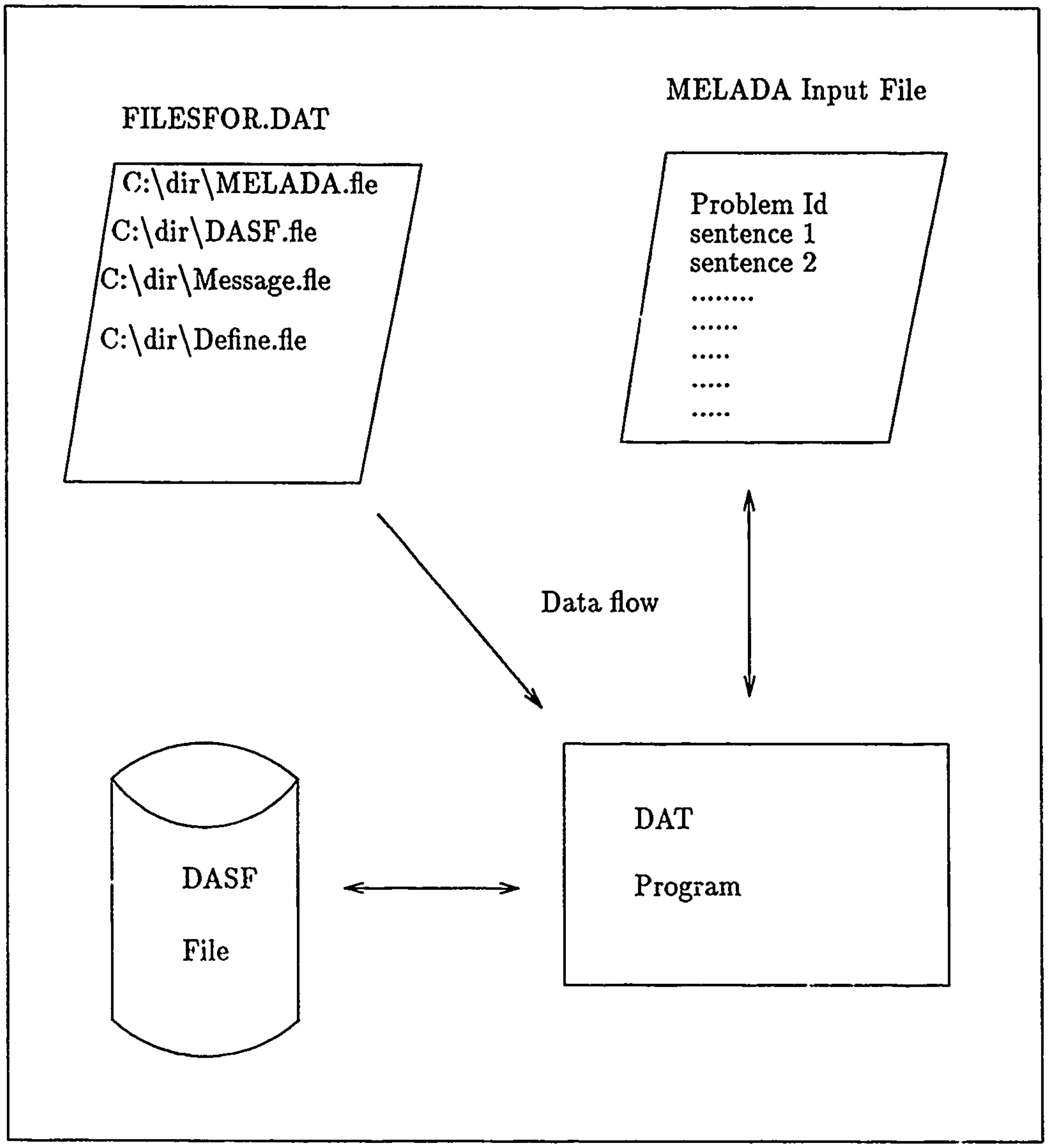


Figure 4.4. DAT Implementation Relationships

V. TREESOLVER: Optimizing Software Using DASF Files

This chapter describes the software designed to meet the third thesis objective of using an optimizing software package with the DASF file. The reason for designing a new software versus using a current package is also discussed.

5.1 Program Philosophy

The TREESOLVER package is a stand alone program, written in Turbo Pascal, designed to work directly on a DASF file to solve and store within the DASF file the optimum results. Because it alters the payoff and optimal alternative fields within the DASF file, making a copy of the DASF file may be advisable prior to solving to permit recovery to an unaltered file. Solving for an optimum is accomplished by obtaining the maximum expected utility (value) of the problem. Thus, if a minimization is desired the payoff values must be entered with their sign reversed.

Besides altering the DASF file, TREESOLVER also provides an output file that lists each decision within the problem, the optimum alternative for the decision, and the payoff of the alternative. The decisions are listed in the order found in the DASF file, not in any logical order or information sequence.

The output file can be named as desired in the file named *filesfor.tre* , a small text file that works exactly like the one used for DAT. The file contains the DASF file name and path on the first line and the desired name and path for the output file on the second line. Figure 5.1 shows the relationship of TREESOLVER and its associated files.

5.2 TREESOLVER vs Current Software Strategy

TREESOLVER was developed rather than building a conversion program to create, for example, an AFids input file, for two reasons. First, current software does

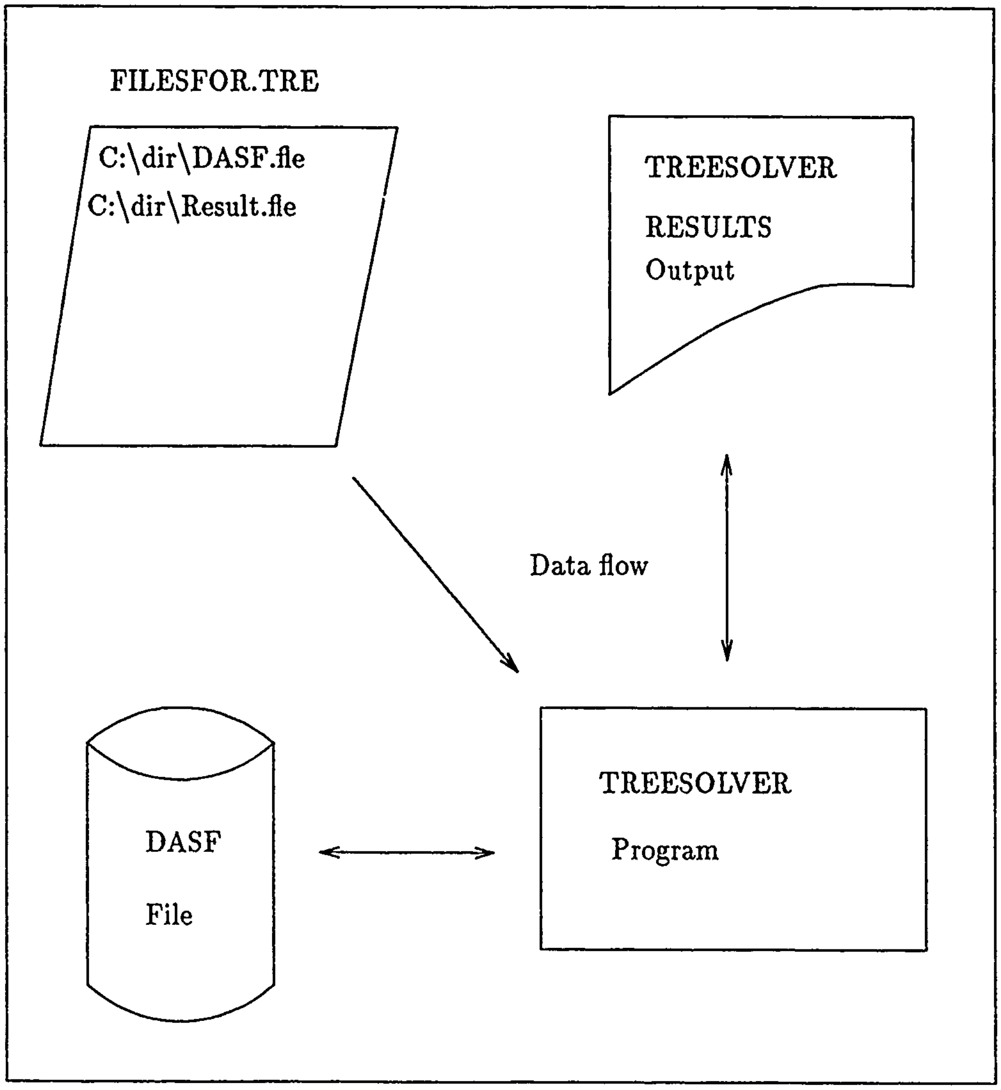


Figure 5.1. TREESOLVER and Associated File Relationships

not have the capability to handle problems needing more storage space than what is available in internal memory. Since DAT is designed to handle large problems without the internal memory limitation, using a current package would reintroduce this limitation to DAT problems. Second, the best optimizer available at AFIT with enough documentation to help in building a conversion package for is AFids. That means the unsolved problem of converting from decision tree to influence diagram would need to be solved. In addition, AFids requires horizontal and vertical positioning information for its graphical representation of the problem which means those coordinates would also need to be generated or input. Thus, it was much easier to develop a new package that would overcome the problem size limitations and yet show that the automated input could be accomplished. Choosing to develop TREESOLVER does not mean that a conversion package for AFids is not possible, only that it was a task deemed too time consuming for this research effort.

VI. Applications and Validation

This chapter provides applications of MELADA, DAT, and TREESOLVER together on three different problems to demonstrate the last objective of this thesis, the viability of having a meta-language and software to standardize and automate the decision analysis field.

A brief problem description and reason for inclusion of each of the three problems is listed below followed by the sections on each individual application.

Umbrella Problem This is the simple problem used to show the difference between decision tree and influence diagram formats. It consists of the decision to take an umbrella or not based on the chance of rain later in the day.

It is used as an example because its size allows for a thorough first inspection of MELADA, DAT, and TREESOLVER in use on a problem that contains the basic elements (decision, alternative, chance, outcome, probability, and payoff) of any decision making problem.

IFF Problem This problem deals with the decision of whether or not to wait to visually identify an incoming aircraft before deciding whether or not to shoot.

This problem was included because it provides a problem larger than the first; it introduces a nonsymmetrical tree structure, decisions between chance nodes, chance nodes conditioned on other chance nodes; and it was provided in story format.

Oil Drilling Problem This problem deals with deciding whether or not to make a seismic test to help determine the possibility of oil prior to deciding whether to drill or lease the land in question.

This problem was selected because it i- a well known standard example problem, can be solved using either influence diagram or decision tree methodology, uses money instead of utility as the payoff, introduces the use of tolls instead of applying the cost to payoffs, and introduces the use of terminal alternatives.

6.1 First Application: Umbrella Problem

The four possible outcomes and the associated payoffs expressed in utils, a nondescript unit of utility, are:

Rain, Umbrella taken : 10 utils

No Rain, Umbrella taken : 4 utils

Rain, No Umbrella : 0 utils (worst case)

No Rain, No Umbrella : 6 utils

The probability of rain for that day is also assumed to be 0.4, making the probability 0.6 for no rain. The decision tree for the problem is given in Figure 6.1.

The MELADA file used for solving the problem, written with extra prose to help comprehension, is listed below.

```
Umbrella Problem
1  The initial decision [take umbrella?] has alternatives
   {yes, no}.
2  Alternative {yes} leads to chance event (rain later?).
3  Alternative [take umbrella?=no] leads to chance event
   (rain?).
4  Chance event (rain?) "is the same chance event as rain
   later? but no umbrella is taken.". 5  The probability of
   rain given an umbrella Pr(rain later?=yes|umbrella) is
   < 0.4>.
6  The outcomes of chance (rain?) are {yes|no umbrella,
   no|no umbrella}. 7 The payoffs for (rain?) are <0,6>.
```

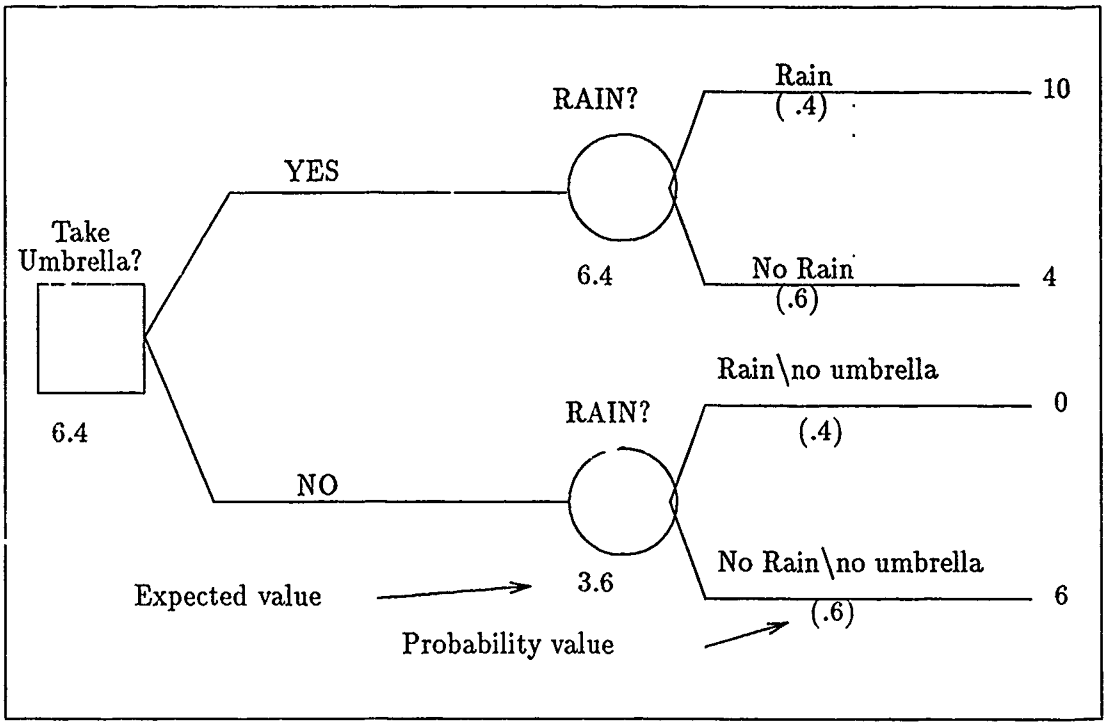


Figure 6.1. Umbrella Problem Tree

```
8  the probability of no rain given an umbrella
   PR(rain later?=no|umbrella) is <.6>.
9  the payoffs for the outcomes of (rain later?) are <10, 4>.
10 The probabilities for the outcomes of Pr(rain?) are <.4,.6>.
11 [take umbrella?] "is the decision on whether or not to
take a- umbrella given the chance for rain later.".
```

Various sentence structures were used to show the flexibility the user has. For example, sentence two shows one way of assigning a successor node to an alternative while sentence three shows the other. Notice that sentence two can only be used after the alternative is assigned to its parent decision node (sentence one). Sentence five is not started on a new line to show the capability of DAT to process the sentences in a paragraph writing style. The last sentence demonstrates the use of the long definition which provides a longer definition or explanation of what the decision take umbrella, mentioned in sentence one, stands for. The long definition will be printed out in a separate definition file to provide the user with a concise listing of selected identifiers and their associated definitions or explanations.

For a comparison of how concise the language can be, the following listing is what the file looks like with only the necessary elements in each sentence to describe the problem. Note that further code reduction could be done by using more compound decision and chance elements to reduce the number of sentences used.

```
Umbrella Problem
1  [take umbrella?] {yes, no}.
2  {yes} (rain later?).
3  [take umbrella?=no] (rain?).
4  (rain?) "is the same chance event as rain
   later? but no umbrella is taken.".
5  Pr(rain later?=yes|umbrella) < 0.4>.
```

```
6  (rain?) {yes|no umbrella, no|no umbrella}.
7  (rain?) <0,6>.
8  PR(rain later?=no|umbrella) <.6>.
9  (rain later?) <10, 4>.
10 Pr(rain?) <.4,.6>.
11 [take umbrella?] "is the decision on whether or not to
take an umbrella given the chance for rain later.".
```

The definition file provided as output by DAT for this problem, which should contain two definitions according to the MELADA input code, is shown below.

```
rain?
is the same chance event as rain later? but no umbrella is taken."

take umbrella?
is the decision on whether or not to take an umbrella given the chance
for rain later."
```

The identifier being defined is placed on the first line with its definition starting on the following line. A maximum of 75 characters per line is printed and a line is skipped between definitions.

The best choice in this case is decided by selecting the alternative with the maximum expected utility. As shown in Figure 6.1 , the expected utility for taking the umbrella is

$$\begin{aligned} U_{umbrella} &= (.4 \times 10) + (.6 \times 4) \\ &= 6.4 \end{aligned}$$

while the expected utility for not taking the umbrella is

$$\begin{aligned} U_{noumbrella} &= (.4 \times 0) + (.6 \times 6) \\ &= 3.6 \end{aligned}$$

Thus, the correct answer is take the umbrella with a payoff of 6.4 utils.

Identical results were obtained by using DAT and TREESOLVER on the MELADA umbrella problem file given above. A printout of the results from TREESOLVER is given in Appendix B.

6.2 Second Application: IFF Problem

The IFF (Identify Friend or Foe) Problem decision tree, with the appropriate probabilities and payoffs included, is given in Figure 6.2.

Description highlights of the problem are given in Appendix B along with the MELADA file derived from the information. Because MELADA requires the correct conditional probabilities for each element, the user must convert the probabilities given using Bayes' Rule to those needed.

The hand calculated results along with the TREESOLVER results are given in Appendix B after the problem description. In this case the TREESOLVER results matched the hand calculations for the problem.

6.3 Third Application: Oil Drilling

This textbook problem is given as an example by Hillier and Lieberman (4:597-613) as well as Baron (2:3). The decision tree for the problem is given in Figure 6.3.

Hillier and Lieberman minimize the expected opportunity loss to solve the problem in their text. But TREESOLVER only maximizes, so the MELADA file was set up to maximize profits. Again, the correct posterior probabilities had to be

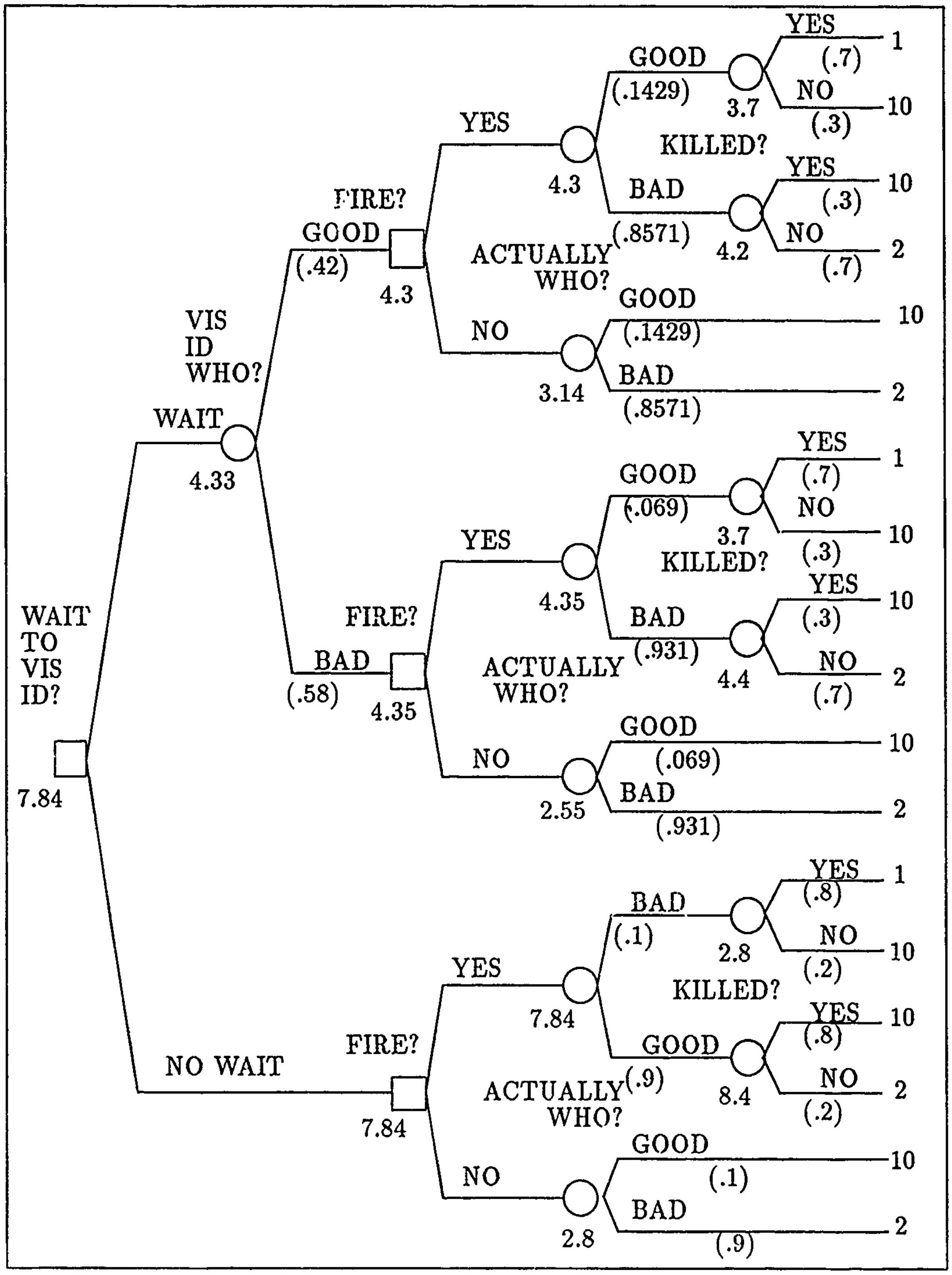


Figure 6.2. IFF Problem Tree

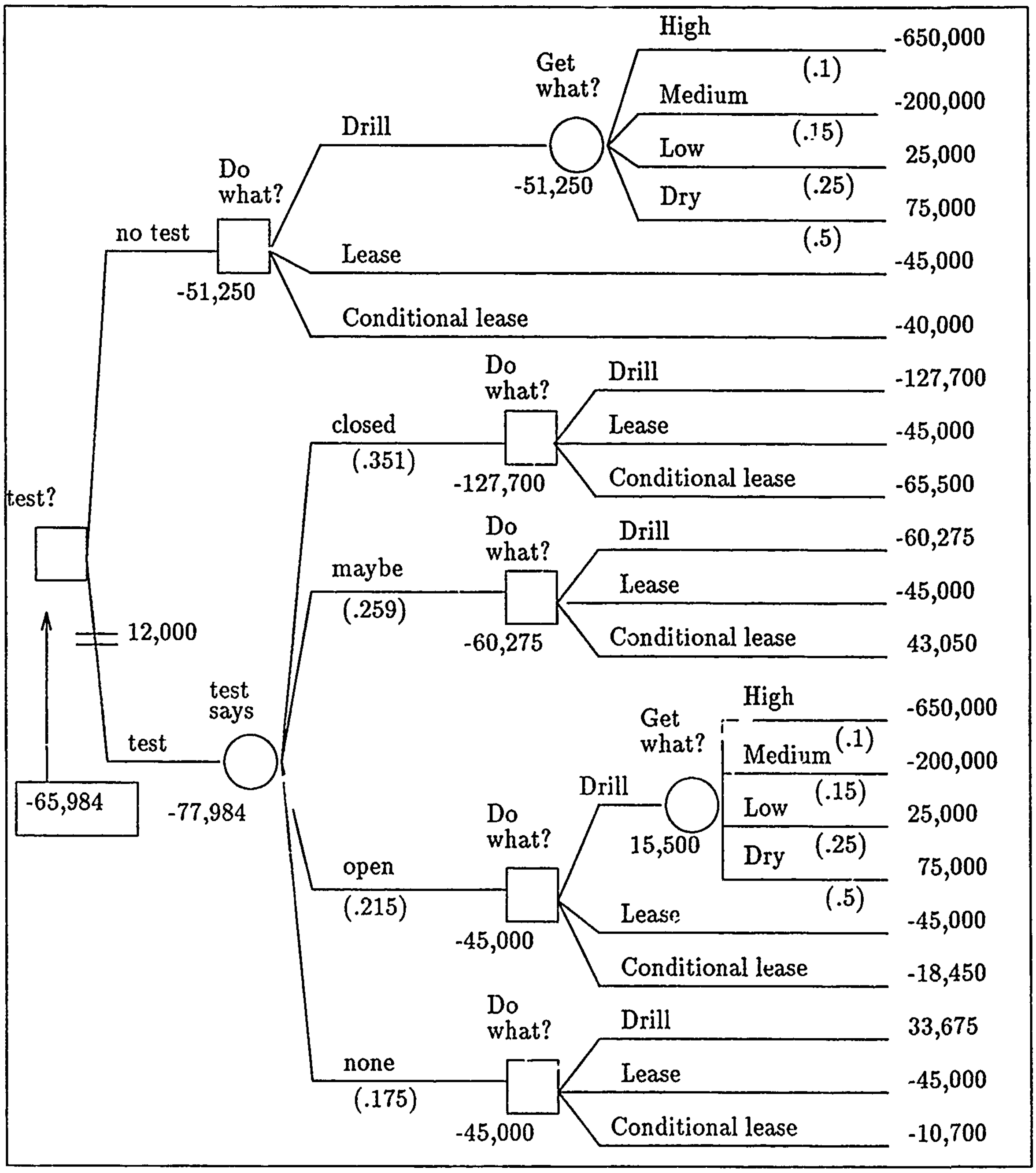


Figure 6.3. Oil Drilling Problem Tree

calculated for input into the file. In this case, the numbers given in the text were used for all probabilities and payoffs (except payoff signs were reversed for maximization) to ensure a common starting ground. This also provided a chance to compare a "proven" solution with the TREESOLVER output.

The TREESOLVER solution nearly matched the text answer. The decisions and payoffs for all nodes were the same except for the overall payoff was different by the cost of the testing, which Hillier and Lieberman apply between the test chance node and the initial decision to test.

6.4 Application Observations

In each case except the last one, the automated solution matched the hand calculations. In the last case, the reason for the difference is MELADA's inability to handle any cost of testing treated as a toll. MELADA can only handle the testing costs by accounting for them in those payoffs conditioned on the testing. Another option, once a conversion program is built for AFids to allow for the use of influence diagrams, is to use a deterministic node to handle the money calculations (2:4) when solving the problem.

Calculations must still be made by the user to provide the correct conditional probabilities for each element. DAT cannot reverse conditional probabilities and does not know if a probability entered is the right one.

Producing the MELADA file goes fairly quick with a word processor if good identifiers are used that allow for copying sentences that are repeated using slightly different identifier names. An example where the sentences were copied and then modified using the find and replace capability of a word processor was in the Oil drilling MELADA input file in Appendix B. Lines 12-21 were copied over to make lines 22-31 and then 'closed' was found and replaced with 'maybe'. Then the probabilities, payoffs, and line numbers were changed as required.

Although not required, numbering sentences and putting each one on a separate line aids in debugging the file. For a very small problem it would be easier to solve by hand than make up the MELADA file. Only when the size becomes comparable to the oil drilling or iff problem does use of the automated system make sense for saving time.

In each case MELADA was able to describe the problem, DAT was able to format the problem in the standard format, and TREESOLVER provided results consistent with hand calculations.

VII. Areas for Further Research and Conclusions

7.1 Areas for Further Research

Since MELADA and DAT are basic prototype attempts to standardize and automate the decision analysis problem description process several future areas for research and enhancement are possible.

The following areas deal with the MELADA language primarily but would require changes or updates to the DAT software.

- Provide for the inclusion of continuous decision and random variables. This would require a research effort to determine what additional concepts would be required. Even influence diagram work is very basic in this area (2:49) and the work involved would probably extend current limits.
- Provide more user work reduction capability such as a way to equate elements that have identical data or nearly identical data.

The next group of topics deal primarily with the software developed during this research effort but may require MELADA changes.

- Addition of a graphics capability. Graphics would allow a viewing of the logical tree structure and its contents. An automated location computation algorithm that represents n -ary trees would need to be researched, otherwise, the DASF file would probably need the inclusion of location coordinates for positioning on the computer screen.
- Develop the conversion program necessary to transfer between tree and influence diagram formats. Additional MELADA constructs may be needed such as using '&' to link decision and chance nodes that can be collapsed into one node for an influence diagram. An additional means for describing deterministic and value nodes may be needed if software other than AFids were to be

used. Like in the graphics enhancement case above, the location computation algorithm would be helpful here.

- Improvement in processing speed. Research into various systems such as a buffer system within internal memory that would store a large number of records. Processing would take place on the records in memory reducing the delays required when reading a file. The file only would be accessed when writing the buffer back and reading in another block that contains the records needed.

- I lusion of a program to reverse the order of conditional probabilities (apply Bayes'Rule) for use when entering the data in a MELADA file. Taken one step further, research into incorporating this capability into DAT to allow for MELADA to provide general probabilities and let DAT provide the correct one for each case.

- Inclusion of a program to provide value lotteries for use in determining payoffs for the MELADA file. Perhaps the program could be called by the DAT program when it finds missing payoff values or at least allow the user to execute it if desired at that point.

- Provide an interactive shell to group the programs together and allow for interactive update of the MELADA file, files containing the input and output file paths and names, and the DASF file. In addition, each program could be run by selection from a menu.

- TREESOLVER could be modified to allow for minimization problems and provide the capability to output the results to several devices at the same time.

7.2 Conclusions

MELADA provides a good, basic taxonomy of decision analysis for problems with noncontinuous random events and decisions. It allows for the use of word processors to build complete problem description files for input into software packages that can then solve the problem. Although at the present time influence diagram optimizing software cannot be used to solve a problem, MELADA has the basic elements in place to provide for that option. In addition, the language has the capacity for expansion of new elements.

The associated software, DAT and TREESOLVER, provide an automated input and solution capability as demonstrated by several example problems. Increased user flexibility and standardization within the decision analysis field can be achieved through the use of the DASF file as an input file for future software packages and for conversion packages for existing software. The software also overcomes the limitation of internal memory on problem size by using the DASF file for working storage rather than internal memory. Because the software is written in Turbo Pascal and contains documentation within the source code, programmers can expand and update the programs easily.

Although not a polished package, the MELADA language and software developed in this research effort provide the ground work for a viable, standardized language and format for decision analysis.

Appendix A. *MELADA, DAT, and TREESOLVER User Manual*

This is a checklist guide designed for the casual user for preparing MELADA files, Running DAT, and Running TREESOLVER.

A.1 MELADA Input File Preparation Checks

- Use a word processor to build an ASCII file containing the MELADA language describing the problem.
- Use the syntax charts given in Figure A.1 and Figure A.2 to ensure proper MELADA element and sentence structure. Instructions for using the charts are:

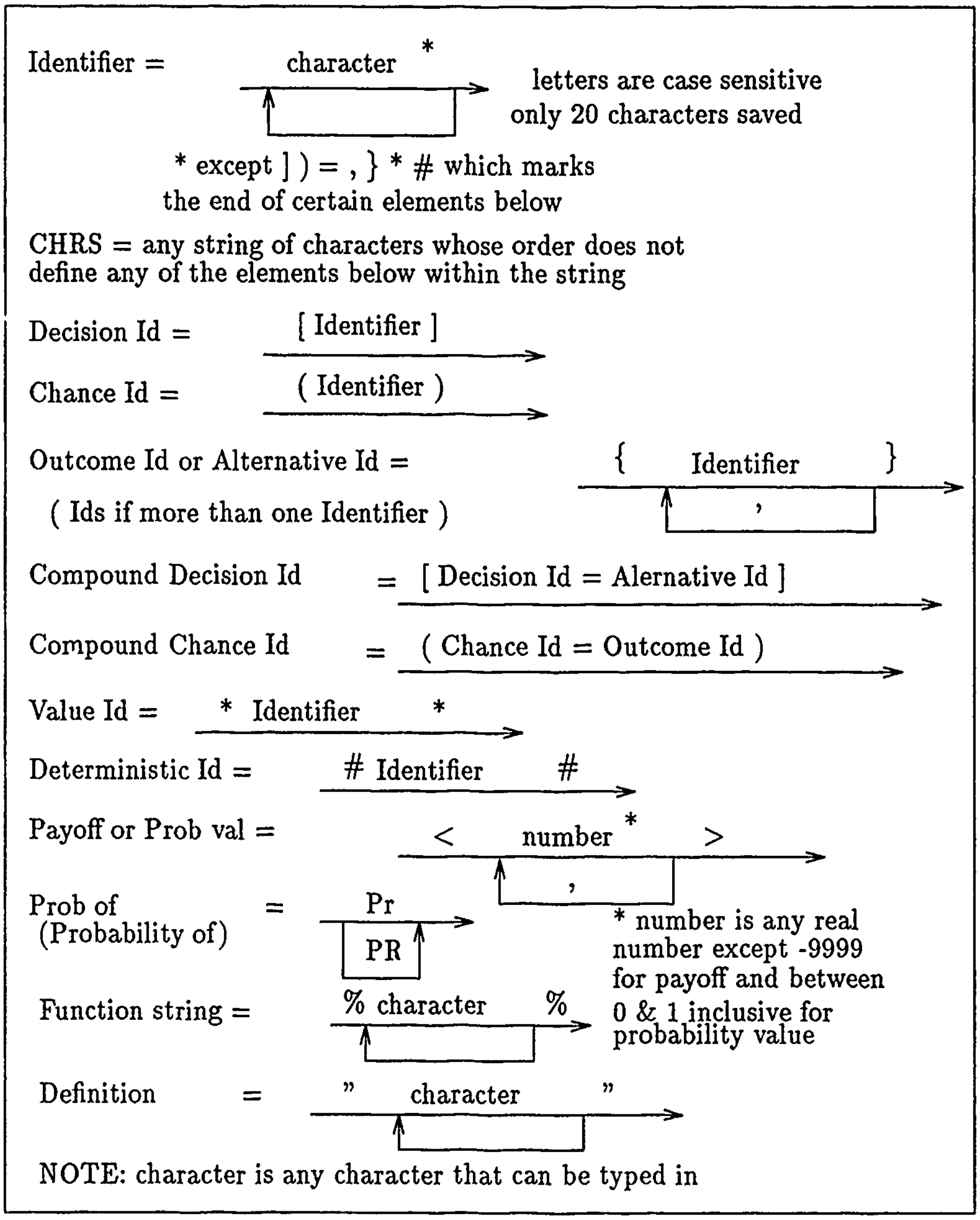
    1. Arrows show direction of progression (order of inputs). General progression is left to right. Complete travel from left to right must be made to ensure including all required items for each element or sentence.
    2. When multiple paths are depicted by arrows going from left to right, any one path or option may be taken. For example, using the element *Prob of* (probability of), shown on the element syntax chart third from the bottom, the option is given to use either **Pr** or **PR** .
    3. Arrows directed from right to left depict the ability to repeat items within the "cycle" created by the arrow as many times as necessary unless restrictions are given. For example, on the MELADA element chart, *Identifier* is made by repeatedly adding a character, but only 20 characters are used.
    4. In the sentence chart, any element within the <> are optional.
    5. Any combination not given by the syntax charts is not a legal MELADA construct and will not be recognized by DAT.

- In addition to having proper syntax, ensure the MELADA input file has:
    1. a problem identifier in the first 20 characters of the first line of the file.
    2. the sentences starting on the second line.
    3. a distinct identifier for each element.
    4. an alternative or outcome linked to its parent decision or chance node prior to the assignment of payoffs or probabilities.
    5. not assigned a payoff to any alternative or outcome that has a successor.
    6. every outcome assigned a probability.
    7. a probability or payoff assigned to an element only once.
- To aid in debugging, begin each sentence on a new line and number the sentence.
- If a sentence exceeds the line length, **DO NOT** break an element in the middle of an identifier, payoff value, or probability value. **ONLY** break an element if it has multiple identifiers, payoffs, or probabilities, and make the break between identifiers (after a comma).
- Remember that MELADA is case sensitive so "Name" and "name" are considered distinct identifiers.

A.2 DAT Preparation Checklist

These guidelines are designed to help in setting DAT up for processing a MELADA file.

- Compile the source code using the Compilation Checklist included at the end of this manual if a DAT.EXE file does not exist.
- Place the DAT.EXE file in a drive other than the one used for the DASF file to allow for increased storage capacity for the DASF file.



letters are case sensitive
only 20 characters saved

* except]) = , } * # which marks the end of certain elements below

CHRS = any string of characters whose order does not define any of the elements below within the string

* number is any real number except -9999 for payoff and between 0 & 1 inclusive for probability value

NOTE: character is any character that can be typed in

Figure A.1. MELADA Syntax Element Structure

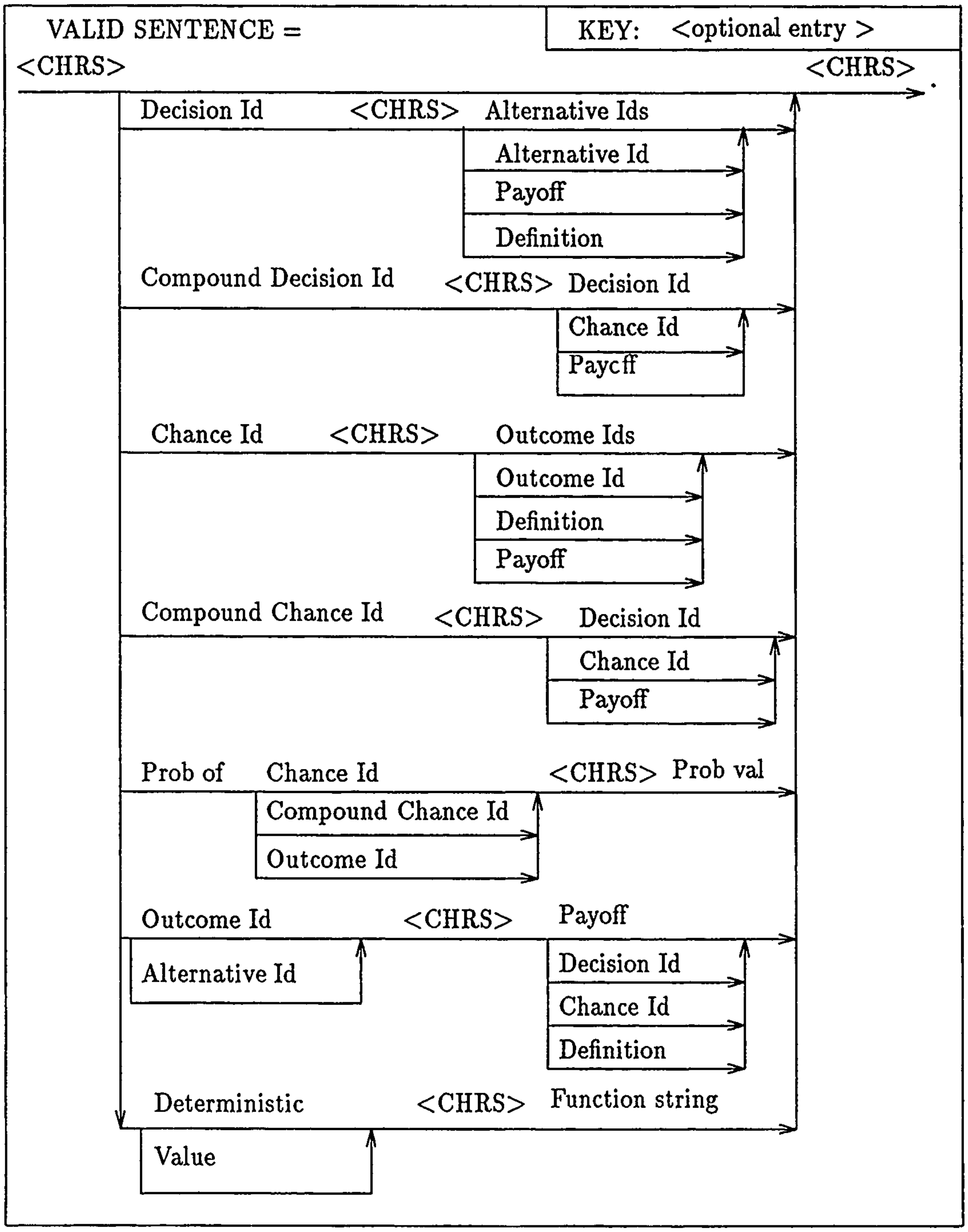


Figure A.2. MELADA Syntax Sentence Structure

- Create a FILESFOR.TRE ASCII file in the same directory the DAT.EXE file is located using the instructions below.

  1. Place the drive, path, and name of the chosen MELADA input file as a single string (`C:\DIRECTORY\FILENAME.EXT`) on line one of the file.
  2. Place desired drive, path, and name of DASF file on line two of the file.
  3. Place desired drive, path, and name of message file on line three of the file.
  4. Place desired drive, path, and name of definition file on line four of the file.

  An example FILESFOR.TRE for a hypothetical computer system with two hard drives, labeled "C" and "D", which will be used to hold all files, is given below. DAT and all the files except the DASF file will be placed in the directory "DA" on drive "C" while the DASF file will be in the root directory of drive "D". In this example, the type of file will be used as the filename and the extension will be .INP for input files and .OUT for output files.

```
C:\DA\MELADA.INP
D:\DASF.OUT
C:\DA\MESSAGE.OUT
C:\DA\DEFINE.OUT
```

  WARNING: the three output files (DASF, Message, and Definition) will overwrite any existing file with the same name designated in this file.

- Run DAT by first ensuring the DOS prompt of the drive and directory containing the DAT file is displayed on the CRT screen and then type:

```
    DAT <cr>
```

  DAT will give real time processing status updates on the CRT screen and inform if errors were detected.

A.3 TREESOLVER Preparation Checklist

This checklist provides guidance for using TREESOLVER to solve a problem stored in a DASF file.

1. Compile source code (TREESOLVER.PAS) if a TREESOLVER.EXE file does not exist. The only compiler option needed is to compile to disk to get an executable file. Further help can be obtained from Turbo Pascal User's Guides.
2. Select a drive and directory to place the executable TREESOLVER file.
3. Create an ASCII file named FILESFOR.TRE using the instructions below and place it in the same directory as the TREESOLVER executable file.
   - Place the drive, path, and filename of the DASF file holding the problem to be solved on the first line exactly the way described for the FILESFOR.DAT file in the DAT Preparation Checklist.
   - Place the drive, path, and filename for the output file that will hold the problem solution on the second line in the same manner as for the DASF input file.
4. Run TREESOLVER by ensuring the current drive and directory is the one containing TREESOLVER, and type at the DOS prompt:
   ```
   TREESOLVER <cr>
   ```
   TREESOLVER provides a running count of decision and chance nodes "solved" during execution on the CRT screen.

WARNING: TREESOLVER makes updates to the DASF file and overwrites any file with the drive, path, and filename given for the output file.

NOTE: Remember that TREESOLVER only maximizes expected utility, thus minimization problems must have the payoffs' signs reversed to provide correct results.

A.4 DAT Compilation Checklist

This checklist provides only general guidance and assumes the user has some familiarity with compiling Turbo Pascal programs. If further help is needed, refer to a Turbo Pascal User's Guide or someone with Turbo Pascal expertise.

1. Ensure the following files are present in one directory:

   DAT.PAS Main program file.

   GLOBALS.PAS Source code of the Pascal Unit containing all global variables and declarations for DAT.

   READSENT.PAS Pascal Unit for the procedure Readsentence, which drives the parsing of each MELADA sentence.

   DATPRO.PAS Pascal Unit containing the procedures used by DAT.

   DATFUN.PAS Pascal Unit containing the functions used by DAT.

2. Set the compile to disk option to obtain compiled files. No other compilation options need be set.

3. Compile DAT using the BUILD command. This will cause the compilation of all the Units along with DAT, including the recompilation of any files already compiled. Compiled Units have the same name as the source code but have .TPU extensions. Use of the BUILD command will ensure an executable DAT file based on the compiling computer's capabilities.

NOTE: Units may also be compiled individually. Compile with the compile to disk option just like for programs. Instead of .EXE files, the .TPU files will be built. After all Units have been compiled, compile the DAT.PAS file with the compile to disk option.

Appendix B. *Application Problem Listings*

In addition to the listings shown here, the MELADA input files for all three problems are included with the DAT and TREESOLVER source code on diskette.

B.1 Umbrella Problem

Since the input files were listed earlier only the TREESOLVER output file will be included here.

```
     Results/Decisions for Umbrella Problem


      DECISION                 BEST ALTERNATIVE          PAYOFF

--------------------     --------------------     ----------


     take umbrella?                         yes       6.400E+00
```

B.2 IFF Problem

The IFF problem story description highlights are listed below.

- Prior odds of an aircraft being hostile were 9 to 1.
- Probability of a kill if fired before visual range was .8.
- Probability of a kill if waiting for visual id and plane was a friendly was .7.
- Probability of a kill if waiting for visual id and plane was enemy was .3.
- Probability of correct visual identification was .6.
- Utility of killing an enemy equals letting a friendly through (live).
- Killing a friendly equals one half the utility of letting the enemy through.
- Letting a friendly through was 5 times the utility of letting an enemy through.

In this situation the utility of letting a friendly through was set at 10. The MELADA file for the problem is listed below.

```
IFF test problem
1   [wait to vis id=wait] (vis id who?).
2   [wait to vis id=no wait] [fire|no wait].
3   [fire|no wait=fire now] (who|fire now).
4   [fire|no wait=no fire now] (who|no fire now).
5   (who|no fire now=good guy) <10>.
6   (who|no fire now=bad guy) <2>.
7   Pr(who|no fire now) <0.1,0.9>.
8   (who|fire now=good|fire now) (kill?|good).
9   (who|fire now=bad|fire now) (kill?|bad).
10  (kill?|good) {killed|good,no kill|good}.
11  (kill?|bad) {killed|bad,no kill|bad}.
12  (kill?|good) <1,10>.
13  (kill?|bad) <10,2>.
14  Pr(who|fire now) <0.1,0.9>.
15  Pr(kill?|good) <.8,.2>.
16  PR(kill?|bad) <.8,.2>.
17  (vis id who?=id good) [fire?|id good].
18  (vis id who?=id bad) [fire?|id bad].
19  Pr(vis id who?) <.42,.58>.
20  [fire?|id good=fire|id good] (who|fire id g).
21  [fire?|id good=no fire|id good] (who|no fire id g).
22  [fire?|id bad=fire|id bad] (who|fire id b).
23  [fire?|id bad=no fire|id bad] (who|no fire id b).
24  (who|fire id g=good|fire id g) (kill?|good id g).
25  (who|fire id g=bad|fire id g) (kill?|bad id g).
```

```
26 (who|no fire id g) {good id g, bad id g}.
27 Pr{good id g} <.1429>.
28 Pr{bad id g} <.8571>.
29 {good id g} <10>.
30 {bad id g} <2>.
31 (kill?|good id g) {killed|good id g,no kill|good id g}.
32 Pr(kill?|good id g) <.7,.3>.
33 (kill?|good id g) <1,10>.
34 (kill?|bad id g) {killed|bad id g, no kill|bad id g}.
35 (kill?|bad id g) <10,2>.
36 Pr(kill?|bad id g) <.3,.7>.
37 (who|fire id b=good|fire id b) (kill?|good id b).
38 (who|fire id b=bad|fire id b) (kill?|bad id b).
39 (who|no fire id b) {good id b, bad id b}.
40 Pr{good id b} <.069>.
41 Pr{bad id b} <.931>.
42 {good id b} <10>.
43 {bad id b} <2>.
44 (kill?|good id b) {killed|good id b,no kill|good id b}.
45 Pr(kill?|good id b) <.7,.3>.
46 (kill?|good id b) <1,10>.
47 (kill?|bad id b) {killed|bad id b, no kill|bad id b}.
48 (kill?|bad id b) <10,2>.
49 Pr(kill?|bad id b) <.3,.7>.
50 Pr(who|fire id b) <.069,.931>.
51 Pr(who|fire id g) <.1429,.8571>.
```

The "approved" results for the decisions involved in the IFF problem are:

Wait to vis Id? no; payoff = 7.84.

Fire—no wait yes; payoff = 7.84.

Fire—id friendly yes: payoff = 4.299.

Fire—id enemy yes; payoff = 4.352.

The results from TREESOLVER are listed below.

```
  Results/Decisions for IFF test problem


       DECISION                BEST ALTERNATIVE          PAYOFF
--------------------      --------------------     ----------


     wait to vis id                    no wait      7.840E+00
       fire|no wait                   fire now      7.840E+00
      fire?|id good              fire|id good      4.300E+00
       fire?|id bad               fire|id bad      4.352E+00
```

B.3 Oil Problem

The MELADA file for the wildcat oil drilling problem is given below.

```
Oil Drilling Problem
1  [test=no][do|no test].
2  [do|no test=drill|no test](well|no test).
3  (well|no test) {hi|no test,med|no test,low|no test,
   dry|no test}.
4  (well|no test) <650000,200000,-25000,-75000>.
5  Pr(well|no test) <.1,.15,.25,.5>.
6  [do|no test=lease|no test] <45000>.
7  [do|no test=cond lease|no test] (well|cond|no test).
8  (well|cond|no test)
```

```
   {hi|cond|no test, med|cond|no test,low|cond|no test,
   dry|cond|no test}.
9  (well|cond|no test) <250000,100000,0,0>.
10 Pr(well|cond|no test) <.1,.15,.25,.5>.
11 [test=yes](test says).

12 (test says=closed)[do|closed].
13 [do|closed=drill|closed](well drilled|closed).
14 (well drilled|closed){hi drilled|closed,
   med drilled|closed, low drilled|closed,
   dry drilled|closed} .
15 Pr(well drilled|closed) <.166,.24,.327,.267>.
16 (well drilled|closed)  <650000,200000,-25000,-75000>.
17 [do|closed=lease|closed] <45000>.
18 [do|closed=cond lease|closed] (well leased|closed).
19 (well leased|closed) {hi leased|closed, med leased|closed,
    low leased|closed, dry leased|closed}.
20 (well leased|closed) <250000,100000,0,0>.
21 Pr(well leased|closed) <.166,.24,.327,.267>.

22 (test says=maybe)[do|maybe].
23 [do|maybe=drill|maybe](well drilled|maybe).
24 (well drilled|maybe){hi drilled|maybe, med drilled|maybe,
   low drilled|maybe, dry drilled|maybe} .
25 Pr(well drilled|maybe) <.129,.108,.241,.522>.
26 (well drilled|maybe)  <650000,200000,-25000,-75000>.
27 [do|maybe=lease|maybe] <45000>.
28 [do|maybe=cond lease|maybe] (well leased|maybe).
```

```
29 (well leased|maybe) {hi leased|maybe, med leased|maybe,
    low leased|maybe, dry leased|maybe}.
30 (well leased|maybe) <250000,100000,0,0>.
31 Pr(well leased|maybe) <.129,.108,.241,.522>.

32 (test says=open)[do|open].
33 [do|open=drill|open](well drilled|open).
34 (well drilled|open){hi drilled|open, med drilled|open,
  low drilled|open, dry drilled|open} .
35 Pr(well drilled|open) <.039,.087,.146,.728>.
36 (well drilled|open)  <650000,200000,-25000,-75000>.
37 [do|open=lease|open] <45000>.
38 [do|open=cond lease|open] (well leased|open).
39 (well leased|open) {hi leased|open, med leased|open,
    low leased|open, dry leased|open}.
40 (well leased|open) <250000,100000,0,0>.
41 Pr(well leased|open) <.039,.087,.146,.728>.

42 (test says=none)[do|none].
43 [do|none=drill|none](well drilled|none).
44 (well drilled|none){hi drilled|none, med drilled|none,
   low drilled|none, dry drilled|none} .
45 Pr(well drilled|none) <0,.107,.238,.655>.
46 (well drilled|none)  <650000,200000,-25000,-75000>.
47 [do|none=lease|none] <45000>.
48 [do|none=cond lease|node] (well leased|none).
49 (well leased|none) {hi leased|none, med leased|none,
    low leased|none, dry leased|none}.
```

```
50 (well leased|none) <250000,100000,0,0>.
51 Pr(well leased|none) <0,.107,.238,.655>.
52 Pr(test says) <.351,.259,.215,.175>.
```

The textbook results (4:611) for the oil drilling problem decisions were:

Do seismic test? yes; payoff = -65,984.

Do what—no test? drill; payoff = -51,250.

Do what—closed drill; payoff = -127,700.

Do what—maybe drill; payoff = -60,275.

Do what—open lease; payoff = -45,000.

Do what—none lease; payoff = -45,000.

Adding the test costs back in will result in the test decision's payoff changing to -77,984, the payoff computed for alternative yes of that decision.

The results from TREESOLVER without taking into account the testing cost tolls are shown below. Remember, the payoff signs are reversed.

```
        Results/Decisions for Oil Drilling Problem


        DECISION             BEST ALTERNATIVE          PAYOFF
 --------------------   --------------------    ----------


                test                     yes     7.798E+04
          do|no test           drill|no test     5.125E+04
           do|closed            drill|closed     1.277E+05
            do|maybe             drill|maybe     6.027E+04
             do|open              lease|open     4.500E+04
             do|none              lease|none     4.500E+04
```

Appendix C. *DAT Procedures and Function Summaries*

This appendix contains brief summaries of what each function and procedure does, the variables involved, and the other functions and procedures called. The functions and procedures are listed in separate sections and listed in alphabetical order within each section.

C.1 Listing of Functions

```
FIND_RECORD( var id : id_str;
       var dat_file : data_file) : longint;

Checks DASF file records for id given.

Input variables:
     id = identifier name to search for
     dat_file = DASF file

Output variables:
     Find_Record = record number of the match
                   -1 if there is no match

Calls: Get_record, reset, filesize
-------------------------------------------------------
```

GET_ID(var F: text; stopchr : string) : string;

Reads in the identifier for an element from the MELADA Input File. Only allows 20 characters to the iaentifier and has a warning message printed if it is exceeded prior to reaching the stop character: storpchr

Input Variables:

F = MELADA Input File

stopchr = one of the special delimiting characters

Output variable:

Get_Id = identifier (name)

CALLS: Warning

GET_PROB(var F : text) : real;

Reads in the probability from the file assinged to F, prints out a warning if the value is not legal.

Input variable:

F = MELADA Input File

Output:

GET_PROB = probability value between 0 and 1

Calls: Get_Id, Val, Str, Warning

```
-------------------------------------------------------

LOOP_ADDED(     node : stprage_node;
            succ_rec : longint) : boolean;

Checks to see if a cycle is added to the records in the
DASF file.  Check is accomplished by seeing if the
successor to be added is already a predecessor further
up the logical tree of DASF records.

Input variables:
     node = record containing the predecessor of the
            node to be added
     succ_rec = record number of the node to be
                added as a successor.

Output variable:
     LOOP_ADDED = True, if there would be a cycle
                  False, if no cycle formed

Calls:  Get_Record, Warning
-------------------------------------------------------
```

C.2 Listing of Procedures

```
ADD_ARC(var pred node : storage_node;
           pred index : longint;
         var arc_node : storage_node;
            arc_index : longint;
         var dat_file : data_file);
```

Links predecessor node (record) with successor node (record). Checks to see if a cycle would be added, the predecessor already has a successor, and if the successor already has a predecessor. Has an error message printed if any of those conditions occur and does not link the nodes.

Input variables:

pred_node = predecessor node
pred_index = DASF record number of predecessor node
arc_node = successor node
arc_index = DASF record number of successor node
dat_file = DASF file

Output variables:

pred_node = gets arc_index put in succ_rec field
arc_node = has pred_index put in pred_rec field

Calls: Loop_Added, Store_Node, Get_Record, Warning
Updates global variable: arc_not_added

```
ASSIGN_VALUES( node : storage_node;
               index : longint);
```

Assigns the payoff values to the various elements dependent upon type. Checks to see if the element has a successor. If it does, a value is not assigned, but an error message is written. Stores node in DASF file at record index after payoff is assigned to node.

Input variables:

```
     node = RAM storage record holding element to be
            assigned payoff
     index = DASF record number for node
```

Output variable:

```
     node = Gets payoff assigned to field payoff
```

Calls: Warning, Get_Record, Get_Value_For, Store_Node,
Uses and changes global variable: Inchr

```
CHECK_FOR_CORRECT_DATA(var dat_file : data_file);
```

Checks the DASF file completeness after all input from the MELADA file. Also assigns record keeping info (root node record number, last record number) to record zero in the DASF file. Prints error messages for any errors found.

Input variable:

    dat_file = DASF file

Calls: Get_Record, Store_Node, Warning, Filesize

```
GET_NODE( var node : storage_node;
         var index : longint;
           stopchr : string;
      type_of_node : node_types);
```

Reads in the name (identifier) of the node and checks to see if it exists already. If it does then it checks to see if it is the correct type, warning message if type is wrong. If id is new then initializes a new node.

Input variables:

    node = RAM storage record whose name is being checked
    index = number of the record with the same name
    stopchr = right delimiting symbol passed to Get_Id
    type_of_node = node type (decision, chance, etc.)

Calls: Get_Id, Find_Record, New_Node, Store_node, Warning

```
GET_RECORD(record_number : longint;
            var dat_file : data_file;
                var node : storage_node);
```

Reads DASF file record into variable node.

Input variables:

     record_number = DASF record to be read

     dat_file = DASF file

     node = a RAM record that holds the data read
            in from a DASF record

Output variable:

     node = storage record filled with data of record
            read in

Calls: Reset, Read, Seek, Close

```
GET_VALUE_FOR(var node : storage_node);
```

Reads in the payoff value from the MELADA Input File and places it it the payoff field of node. Writes an error message if the node already has a payoff value and does not update the payoff value.

Input variable:

    node = RAM storage record exactly like DASF record

Output variable:

    node = payoff value in field payoff

Calls: Get_Id, Val, Str, Warning

PROCEDURE LONGDEF(var Id : Id_str);

Reads in long definition for Id from MELADA Input File and writes Id and the definition to the output file assigned to hold definitions.

Input variable:

    Id = MELADA element Identifier (Name)

Calls: Read, Writeln, Append, Close

Uses global variable: Inchr

```
NEW_NODE(var node : storage_node;
     var file_num : longint);
```

Initializes a storage node with the name given by the input node and is assigned the record position file_num for the DASF (data file) file.

```
Input variables:
     node = temporary storage record being initialized
              to hold data of new MELADA element (arc or
               node)
     file_num = record position in DASF file for node

Output variable:
     node = the input node fully initialized

Uses and increments global variable next_record
--------------------------------------------------------


READ_SENTENCE;

Main procedure that parses one sentence in the MELADA
file by finding the key elements based on key delimiting
characters and the order found.

CALLS: Read, Eof, Get_Id, Find_Record, Get_Record,
       Warning, Get_Node, Add_Arc, Store_Node, Get_Prob,
       Longdef, Assign_Values

Changes global variable:  Inchr
--------------------------------------------------------
```

```
STORE_NODE(var node : storage_node;
              index : longint;
       var dat_file : data_file);

Stores a node in a DASF record.

Input variables:
     node = RAM record containing data to be stored
     index = record number in DASF file to store data at
     dat_file = DASF file

Calls : Seek, Write, Reset, Close
--------------------------------------------------------


WARNING(num : warn_num; id : Id_str);

prints error message to a file assigned to msg
Input variables:

     num = number of warning statement to be printed
     id = name or identifier of node with the error

uses global variable:  sent_num
changes global variable: error_msg}
--------------------------------------------------------
```

Bibliography

1. Anderson, David R. and others. *An Introduction to Management Science* (Fourth Edition). St. Paul: West Publishing Company, 1985.
2. Baron, Capt Christopher T. *Influence Diagrams: Automated Analysis With Dynamic Programming.* MS thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1988. AFIT/GOR/MA/88D-1.
3. Fourer, Robert and others. "A Modeling Language for Mathematical Programming," *Management Science*, *36*(5):519–554 (May 1990).
4. Hillier, Fredrick and Gerald Lieberman. *Operations Research* (Second Edition). San Fransico: Holden-Day, Inc., 1974.
5. Howard, Ronald A. "The Foundations of Decision Analysis." In Howard, R. A. and J. E. Matheson, editors, *Readings on the Principles and Applications of Decision Analysis*, Menlo Park, CA: Strategic Decisions Group, 1984. reprinted from IEEE Transactions on Systems Science and Cybernetics, Vol. SSC-4, No. 3, September 1968.
6. Howard, Ronald A. and James E. Matheson. "Influence Diagrams." In Howard, R. A. and J. E. Matheson, editors, *Readings on the Principles and Applications of Decision Analysis*, Menlo Park, CA: Strategic Decisions Group, 1984.
7. Ligeza, Antoni. "Expert Systems Approach to Decision Support," *European Journal of Operational Research*, *37*(1):100–110 (October 1988).
8. Lindley, Dennis. "Scoring Rules and the Inevitability of Probability," *International Statistical Review*, *50*:1–26 (1982).
9. McNamee, Peter and John Celona. *Decision analysis for the professional with Supertree*. Redwood City, CA: Scientific Press, 1987.
10. Morris, William T. *Decision Analysis.* Columbus, OH: Grid, Inc., 1977.
11. Moskowitz, Herbert and Gordon P. Wright. *Operations Research Techniques for Management*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1979.
12. Owen, Daniel L. "The Use of Influence Diagrams in Structuring Complex Decision Problems." In Howard, R. A. and J. E. Matheson, editors, *Readings on the Principles and Applications of Decision Analysis*, Menlo Park, CA: Strategic Decisions Group, 1984.
13. Pratt, John W. and others. "The Foundations of Decisions Under Uncertainty; An Elementary Exposition," *Journal of the American Statistical Association*, pages 35–57 (June 1964).

14. Raiffa, Howard. *Decision Analysis Introductory Lectures on Choices under Uncertainty.* Menlo Park, CA: Addison-Wesley, 1968.

15. Schoemaker, Paul J. H. "The Expected Utility Model: Its Variants, Purposes, Evidence, and Limitations," *Journal of Economic Literature*, *XX*:529–563 (June 1982).

16. Shachter, Ross D. "Probabilistic Inference and Influence Diagrams," *Operations Research*, *36*:589–604 (July/August 1988).

17. Texas Instruments Inc. *Arborist Decision Tree Software User's Guide*, 1985.